Planning for Hybrid Systems via Satisfiability Modulo Theories

Michael Cashmore MICHAEL.CASHMORE@STRATH.AC.UK
University of Strathclyde
26 Richmond Street, G1 1XH, Glasgow, UK

Daniele Magazzeni DANIELE.MAGAZZENI@KCL.AC.UK
Parisa Zehtabi PARISA.ZEHTABI@KCL.AC.UK
King's College London
Bush House, WC2B 4BG, London, UK

Abstract

Planning for hybrid systems is important for dealing with real-world applications, and PDDL+ supports this representation of domains with mixed discrete and continuous dynamics. In this paper we present a new approach for planning for hybrid systems, based on encoding the planning problem as a Satisfiability Modulo Theories (SMT) formula. This is the first SMT encoding that can handle the whole set of PDDL+ features (including processes and events), and is implemented in the planner SMTPlan. SMTPlan not only covers the full semantics of PDDL+, but can also deal with non-linear polynomial continuous change without discretization. This allows it to generate plans with non-linear dynamics that are correct-by-construction. The encoding is based on the notion of happenings, and can be applied on domains with nonlinear continuous change. We describe the encoding in detail and provide in-depth examples. We apply this encoding in an iterative deepening planning algorithm. Experimental results show that the approach dramatically outperforms existing work in finding plans for PDDL+ problems. We also present experiments which explore the performance of the proposed approach on temporal planning problems, showing that the scalability of the approach is limited by the size of the discrete search space. We further extend the encoding to include planning with control parameters. The extended encoding allows the definition of actions to include infinite domain parameters, called control parameters. We present experiments on a set of problems with control parameters to demonstrate the positive effect they provide to the approach of planning via SMT.

1. Introduction

Planning for hybrid systems is an important area in planning, mainly motivated by the need to deal with real-world applications. Hybrid systems are systems described by discrete as-well-as continuous variables. Many real world problems have hybrid dynamics, subject to (continuous) physical effects and controlled by (discrete) digital equipment. PDDL+ (Fox & Long, 2006) is the extension of Planning Domain Definition Language (PDDL) designed to model hybrid systems,through the use of continuous processes and events.

Examples of real-world domains, where planning for hybrid systems is required, include *unit commitment*, which is a fundamental problem in power systems engineering. Unit commitment is the problem of deciding which generating units should be switched on, and when to switch them on, in order to efficiently meet anticipated demand. The hybrid dynamics is due to the continuous changing cost of energy being produced. It has traditionally been solved as a Mixed Integer Programming (MIP) problem, however Campion et al. (2013) investigate the benefits of using planning over the current established methods. Another planning application involving hybrid systems is investigated by Fox et al. (2011, 2012), in which improving the efficiency of multiple battery usage has been modelled as planning problem. Here PDDL+ processes are needed to model the continuous nonlinear change of level of charge in each battery. Aylett et al. (1998) discussed providing correct planning facilities for a plant, focusing on dealing with flows of chemicals, and propose the use and integration of a special planner for valve sequencing. Williams et al. (2005) considered the challenges of extending plan execution to under-actuated systems that are controlled indirectly through the setting of continuous state variables. Vallati et al. (2016) focus on using mixed discrete-continuous planning to deal

with unexpected circumstances in *urban traffic control*, and use PDDL+ processes to model continuous flow of vehicles.

In all these examples, however, domain dependent approaches are used, as planning for hybrid systems is difficult. Indeed, many efforts are being made to extend current planning systems. As an example, Della Penna et al. (2012) present the planner UPMurphi, a universal planner that is capable of reasoning with mixed discrete/continuous domains while respecting the semantics of PDDL+. That approach, however, relies on discretisation. A number of other approaches have been proposed that can handle *subsets* of PDDL+, as described in Section 8.2.

In this paper we propose a new approach for PDDL+ planning that can handle the whole set of PDDL+ features and respects the full PDDL+ semantics. We propose a Satisfiability Modulo Theories (SMT) encoding of PDDL+ problems based on Boolean Satisfiability Problem (SAT) encodings of classical planning problems (H. Kautz & Selman, 1996; Rintanen, 2010b). Planning as SAT has been effective for classical planning problems (e.g. the planner SATPlan (H. A. Kautz, Selman, & Hoffmann, 2006)) and through carefully devised preprocessing and encoding can be applied effectively to temporal planning problems (e.g. the planner ITSAT (Rankooh & Ghassem-Sani, 2015)). In contrast to the state-based search approach to planning, these planners perform a search on encodings of a plan trace, iteratively deepening the trace's horizon. We build on this previous work, using the additional expressive power of SMT to deal with non-linear polynomial dynamics in a planner that fully supports PDDL+ planning for real-world hybrid systems.

Moreover, the proposed encoding also allows the easy modelling of *Control Parameters* (Savas, Fox, Long, & Magazzeni, 2016), which is an interesting emerging area in planning. Using control parameters allows the modelling of actions with infinite domain parameters (e.g. real numbers). This is a challenging proposition for state-based search planners, as the branching factor becomes similarly infinite. We show that for some domains, extending PDDL2.1 with control parameters actually improves the performance of our SMT-based planner, allowing it to scale to problems that without control parameters cannot be solved in a reasonable time.

The paper is structured as follows. In the next section we provide the background about hybrid systems, PDDL+, planning as SAT, and SMT. In Section 3, we introduce a working example that highlights PDDL+ features. We describe our encoding of PDDL+ into SMT in Section 4, followed by an in-depth example of our encoding using the working example in Section 5. In Section 6 we show how the encoding is extended to describe control parameters, and provide another in-depth example of the encoding. The approach is then evaluated on a set of benchmark problems in Section 7. In Section 8 we give an overview of related works and conclude in Section 9.

2. Background

In this section, we introduce the theoretical foundations of hybrid systems, namely the hybrid automata. We then give a brief introduction to PDDL2.1 and then explain how it is extended to describe the hybrid automata in PDDL+. After this we introduce planning as Boolean Satisfiability and then SMT.

2.1 Hybrid Systems

A hybrid system is a system where there are both continuous variables and discrete logical modes of operation. It represents a powerful model to describe the dynamic behaviour of modern engineering artefacts.

The theory of hybrid automata, introduced by Henzinger (Henzinger, 1996), represents a well-defined formalism for describing hybrid systems. Intuitively, hybrid automata are finite state automata extended with continuous variables that evolve over time. More formally, we have the following:

Definition 1 (Hybrid Automaton). A *hybrid automaton* is a tuple $\mathcal{H} = (Loc, Var, Init, Flow, Trans, I)$, where

- Loc is a finite set of locations,
- $Var = \{x_1, \ldots, x_n\}$ is a set of real-valued variables,

- $Init(\ell) \subseteq \mathbb{R}^n$ is the set of initial values for $x_1, \ldots, x_n$ for all locations ℓ.
- For each location ℓ, $Flow(\ell)$ is a relation over the variables in Var and their derivatives of the form:

$$\dot{x}(t) = Ax(t) + u(t), u(t) \in \mathcal{U},$$

  where $x(t) \in \mathbb{R}^n$, A is a real-valued $n \times n$ matrix and $\mathcal{U} \subseteq \mathbb{R}^n$ is a closed and bounded convex set.
- $Trans$ is a set of discrete transitions. A discrete transition $t \in Trans$ is defined as a tuple (ℓ,g,ξ,ℓ') where ℓ and ℓ' are the source and the target locations, respectively, g is the guard of t (given as a linear constraint), and ξ is the update of t (given by an affine mapping).
- $I(\ell) \subseteq \mathbb{R}^n$ is an invariant for all locations ℓ.

An example is the hybrid automaton for a thermostat shown in Figure 1. Here, the temperature is represented by the continuous variable x. In the discrete location corresponding to the heater being off, the temperature falls according to the flow condition $\dot{x} = -0.1x$, while, when the heater is on, the temperature increases according to the flow condition $\dot{x} = 5 - 0.1x$. The discrete transitions state that the heater *may* be switched on when the temperature falls below 19 degrees, and switched off when the temperature is greater than 21 degrees. Finally, the invariants state that the heater can be on (off) *only* if the temperature is not greater than 22 degrees (not less than 18 degrees).

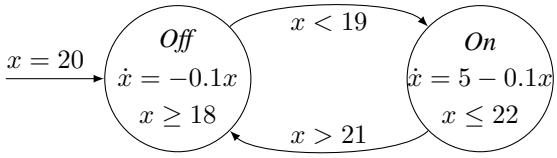


Figure 1: Thermostat hybrid automaton

2.2 PDDL2.1 and PDDL+ Planning

Use of planners in real world problems has challenged the limitations of PDDL and increased the necessity of dealing with time and resources. This lead to the development of PDDL2.1 (Fox & Long, 2003) as the extension of PDDL able to model temporal domains, and allowing planners to reason with continuous numeric change (e.g., (A. J. Coles, Coles, Fox, & Long, 2012; Coles, Amanda and Fox, M and Long, D, 2013)). PDDL2.1 has been extended to PDDL+ (Fox & Long, 2006) to enable the modelling of mixed discrete-continuous domains. In this section we first discuss the semantics of PDDL2.1 and later describe the extensions that have been added in PDDL+.

Definition 2 (PDDL2.1 Planning). A PDDL2.1 planning problem is a tuple $\Pi := \{P,V,A,I,G\}$, where P is a set of propositions; V is a vector of real variables, called fluents; both are manipulated by A, a set of durative and instantaneous actions. $I(P,V)$ is a function over $P \cup V$ which describes the *initial state* of the problem and $G(P,V)$ is a function that describes the *goal condition*.

A durative action $a \in A$ is described as a tuple:

$$a := \{pre_a, \mathit{eff}_a, dur_a\}$$

where $pre_a(P,V)$ is a function over $P \cup V$ that represents the action's preconditions – conditions that must hold for the action to be applied. Similarly, eff_a represents the action's effects, and dur_a is a duration constraint, a conjunction of numeric constraints corresponding to the duration of the action a.

A single condition is either a single proposition $p \in P$, its negation, or a numeric constraint over V. A precondition is a conjunction of zero or more single conditions. The precondition of an action pre_a can be separated into three disjoint subsets:

$$pre_{\vdash a}, pre_{\leftrightarrow a}, pre_{\dashv a} \subseteq pre_a$$

These represent the conditions that must hold at the start of the action, throughout its execution, and at the end of the action, respectively.

Action effects are described by seven subsets:

$$\begin{array}{l} \mathit{eff}^{+}_{\vdash a}, \mathit{eff}^{-}_{\vdash a}, \mathit{eff}^{num}_{\vdash a}, \\ \mathit{eff}^{+}_{\dashv a}, \mathit{eff}^{-}_{\dashv a}, \mathit{eff}^{num}_{\dashv a}, \\ \mathit{eff}_{\leftrightarrow a} \end{array}$$

The first six are the instantaneous effects of adding or removing propositions, or instantaneous numeric effects. These are bound to the start or end of the action. For example, $\mathit{eff}^{+}_{\vdash a}$ denotes the propositions added at the start of the action. In the semantics of PDDL2.1, the values of such instantaneous effects can be exploited to support other actions only after a small amount of time ε (Fox & Long, 2003). This is referred to as *epsilon separation*. The final set, $\mathit{eff}_{\leftrightarrow a}$ is a conjunction of *continuous* numeric effects $\mathit{eff}_{\leftrightarrow}$, which are applied continuously while the action is executing. As a special case, *instantaneous* actions have duration 0, have only one set of preconditions pre_a; and three sets of effects eff^{+}_{a}, eff^{-}_{a}, and eff^{num}_{a}.

Actions can only be applied together at the same time if they are not mutually exclusive (mutex). Actions $a1$ and $a2$ can be applied simultaneously if:

$$\begin{array}{c} pre_{a1} \cap (\mathit{eff}^{+}_{a2} \cup \mathit{eff}^{-}_{a2} \cup \mathit{eff}^{num}_{a2}) = \emptyset \\ \mathit{eff}^{+}_{a1} \cap \mathit{eff}^{-}_{a2} = \mathit{eff}^{+}_{a2} \cap \mathit{eff}^{-}_{a1} = \emptyset \\ \{v1 \in \mathit{eff}^{num}_{a1}\} \cap \{v2 \in \mathit{eff}^{num}_{a2}\} = \emptyset \end{array}$$

PDDL+ is an extension of PDDL2.1, based on hybrid automata semantics. PDDL+ extends PDDL2.1 to support the modelling of exogenous events, reflecting changes that are initiated by the environment. These are introduced by the new constructs of *processes* and *events*.

Definition 3 (PDDL+ Planning). A PDDL+ planning problem is a tuple $\Pi+ := \langle P,V,A,Ps,E,I,G \rangle$, in which P is a set of propositions; V is a vector of real variables, called fluents; and A is a set of durative and instantaneous actions. Ps is a set of processes, and E a set of events. $I(P,V)$ and $G(P,V)$ represent the initial state and goal condition respectively.

The elements P,V,A,I, and G are as in Definition 2. As an analogue, an event $e \in E$ is akin to an instantaneous action: if an event's preconditions pre_e are satisfied, it occurs, yielding the event's instantaneous effects. Similarly, processes are akin to durative actions which apply continuous effects over a period of time.

Note that continuous processes are triggered as soon as their precondition becomes true, and in this sense they *must* be triggered. Exogenous events follow the same semantics. The rationale behind this is that processes and events are used to model changes that are initiated by changes in the world, therefore they are not under the direct control of the executive and are triggered immediately (see Bogomolov et al. (2014) for more details). This is a critical distinction between processes/events and actions. The process/event will automatically occur as soon as its precondition is satisfied; whereas an action will only happen if chosen to be executed by the planner. This is necessary to model exogenous events, (e.g., (Gerevini, Saetti, & Serina, 2006)).

Furthermore, effects become instantaneously available to events, without the epsilon separation. This means, if one event $e1$ is triggered, with effects that satisfy another event $e2$ and trigger a process $p1$, then $e1$, $e2$, and the start of $p1$ all happen at exactly the same time-point. It is due to this behaviour that the semantics of PDDL+ used for plan validation places a bound on the number of cascading (parallel) events – a causal chain of events cannot include a cycle, and a single event cannot be triggered more than once in a single instant (Fox & Long, 2002; Howey, Long, & Fox, 2004).

Given the PDDL+ semantics for modelling hybrid systems, finding an efficient planner that can handle events and processes is the next crucial step. A number of methods have been proposed to handle mixed discrete and continuous domains. We review these in Section 8.

2.3 Planning as Satisfiability

The problem of determining Boolean Satisfiability Problem (SAT) is the problem of finding an assignment of truth values to variables in order to make a set of propositional formulae true. The problem is, given a Boolean expression ϕ with variables $X : \{x_1, x_2, ..., x_n\}$, find an assignment to the variables X that satisfies ϕ, or show that one does not exist. For example,

$$\phi := (\neg x_1 \vee \neg x_2) \wedge (x_1 \vee x_3) \wedge (x_3 \vee x_2) \wedge \neg x_3$$

is false, since there are no assignments to the three variables x_1, x_2, x_3 that will satisfy ϕ. In the context of planning, we are focusing on SAT solvers that solve the decision problem and if the expression is satisfiable, return an example satisfying assignment. Kautz and Selman (H. Kautz & Selman, 1992) formalised propositional planning as a Boolean Satisfiability (SAT) problem in the following way.

First, the classical planning problem for a fixed horizon n is modelled by defining n copies of the Boolean variables which describe the state and actions, and unrolling n times the transition relation that holds between states. This gives a set of variables (X) and constraints between them (ϕ). A satisfying assignment to the variables corresponds to a plan trace.

In more detail, the set of variables X comprises one Boolean variable for each $a \in A$, and each $p \in P$ [1] at each time-step $t_0, \ldots, t_n$. In propositional planning, there are no numeric variables $v \in V$. Constraints are added to ϕ, asserting:

- The initial state holds at time-step 0, and goals at time-step n.
- If an action is true, then its preconditions hold in that time-step.
- If an action is true, then its effects hold in the following time-step.
- Two actions that are mutex cannot be applied together in the same time-step.

To solve the original planning problem without a fixed time horizon, an initial encoding with horizon n_0 is solved, and if found, the satisfying assignment is converted into a corresponding plan. Otherwise, the horizon is incremented and the process is repeated (H. Kautz & Selman, 1999).

A significant number of works have been devoted to formalising planning problems using propositional logic, and improving those encodings (Rintanen, Heljanko, & Niemelä, 2006; Chen, Xing, & Zhang, 2007; Nathan Robinson and Charles Gretton and Duc Nghia Pham and Abdul Sattar, 2009; Rintanen, 2010b; Cashmore, Fox, & Giunchiglia, 2012), which we discuss in Section 8.

2.4 Satisfiability Modulo Theories

Satisfiability Modulo Theories (SMT) (Barrett & Tinelli, 2018) is the problem of deciding the satisfiability of a first-order formula expressed in a given theory. An SMT problem is given as a set of variables and a set of constraints over those variables. In contrast to the SAT problem, the variables are not restricted to Boolean values, but depend upon a *theory*, and the constraints are expressed with respect to a background *logic*. The theory and logic are critical elements of an SMT problem. Theories exist for Boolean propositions, finite and infinite trees, lists, bitvectors, arrays, integers, and reals. SMT takes advantage of the fact that some of them already have efficient solvers in order to reason more efficiently. In our work we use the theory of real numbers and the logic of *quantifier-free non-linear real arithmetic*.

For example, an SMT problem in *quantifier-free linear real arithmetic* might be:

1. Similar to Definition 2, in classical planning we have a set of actions and a set of propositions which are shown as A and P, respectively.

$$\exists u,v,x,y,z \quad \phi$$

where

$$\phi := (x + 3 <= 2u) \vee (v + 4 >= y) \vee (x + y + z >= 2)$$

Problems in SMT can be expressed in the SMT-LIB standard syntax (Barrett, Stump, & Tinelli, 2010). Figure 2 illustrates the example problem above in the standard sytax. The encoding we describe in this paper is produced in this standard format, and can be read many SMT solvers, including Z3 (De Moura & Bjørner, 2008), which we use to solve our problems in Section 7. The encoding also builds upon the prior work discussed in Section 8. Using the same fundamental principles in SMT (using variables of the formula to describe the moments of discrete change) opens the door to applying these works to the PDDL+ setting.

```
( set-logic QF_LRA )
( declare-fun u () Real)
( declare-fun v () Real)
( declare-fun x () Real)
( declare-fun y () Real)
( declare-fun z () Real)
( assert
    (or
    ( <= (+ x 3) (* 2 u ) )
    ( >= (+ v 4) y )
    ( >= (+ x y z ) 2)
    )
)
```

Figure 2: SMT problem in the SMT-LIB standard syntax.

3. Working Example

For the purpose of clarity and ease, we have chosen a simple *Free Fall* problem[2] as our working example and we use this problem to indicate different concepts explained throughout this paper.

In the first part of this section we explain the *Free Fall* problem, and later we describe the hybrid automaton corresponding to this problem. Finally, we provide the PDDL+ model of our problem and map the hybrid automaton to the PDDL+ model. We will then present the SMT encoding of this problem in Section 5.3.

3.1 Free Fall Problem

Our working example considers a ball that falls vertically under the influence of gravity (g) and bounces when it touches the floor. The goal is for a robotic hand to catch the ball after a certain number of bounces. Specific conditions are defined for the robot to determine when the ball can be caught (i.e. within a specific height difference), as shown in Figure 3. Note that the motion is idealised and other forces and their effects (i.e. air resistance) are assumed to be negligible (so the ball will bounce an infinite number of times and every time reach to its initial height). This simplifies the example. The ball is released with an initial velocity v_0, which can be zero ($v_0 = 0$).

3.2 Hybrid Automaton: Free Fall Problem

The hybrid automaton of the Free Fall problem is shown in Figure 4. In this problem the height and velocity of the ball are represented by the continuous variables h and v, respectively. The acceleration of the ball due

2. This example was designed by Derek Long and presented in the tutorial on Planning in Hybrid Domains at ICAPS 2013.

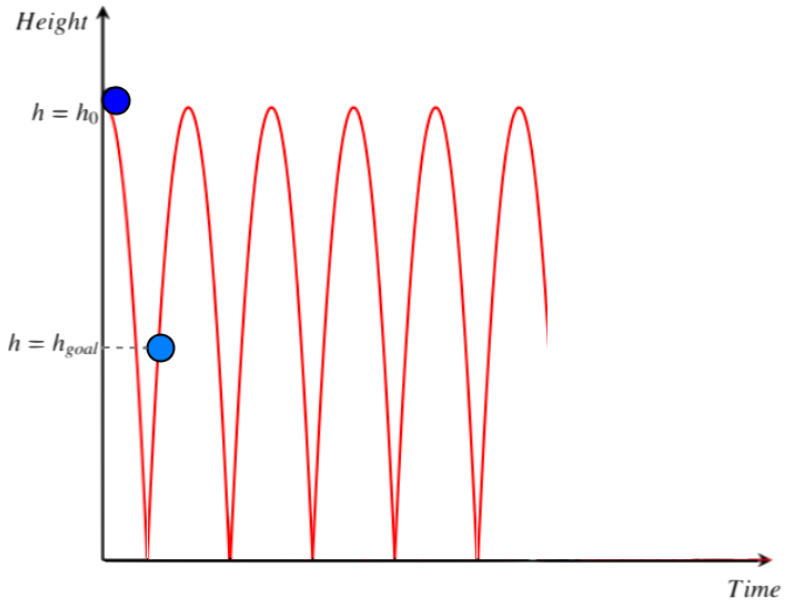


Figure 3: Working example (Free Fall problem): indicates a ball that is released vertically from the initial height of h_0 and bounces as soon as it reaches the floor. The diagram shows a specific height (h_{goal}) at which the robot is able to catch the ball.

to gravity is represented by the variable a. Considering the dynamics of the problem, we have four discrete locations. Note that numeric variables with zero flow have been omitted from locations in Figure 4. Similarly, variables omitted from discrete transitions simply retain their original value. Also, all the conditions and effects related to each discrete transition have been written in the conjunctive form.

The initial state of the automaton is shown on the left side of Figure 4. The ball is held at the height h_0 and the velocity $v = v_0$. As soon as the ball is released, a discrete transition moves the automaton to the next discrete location representing the moving state (this state is shown by the middle circle in Figure 4). In this state, the height of the ball changes according to the velocity $\dot{h} = v$, and the velocity changes according to acceleration $\dot{v} = a$. The whole process of ball's movement is modelled in the $Moving$ location of automaton. A self-looping transition, $Bounce$ negates the velocity of the ball and increases the $number\_bounces$ by one as soon as the ball reaches the ground ($h <= 0.001 \wedge v < 0$). $Bounce$ should happen as soon as its condition is satisfied, or in other words should obey the event's *must* semantics. For this reason the negation of the condition is an invariant of the $Moving$ location. The $Bounce$ transition must happen if and only if the ball reaches the ground ($h <= 0.001$) while moving downward ($v < 0$).

The robot can catch the ball at any point that the ball is at the height of the robotic arm, specified as the interval $[h_{goal}, h_{goal} + 0.1]$ in Figure 4. The goal condition specifies that the ball must be caught after it has bounced at least once. If the ball is caught within the specified height interval and the number of bounces is zero, the hybrid automaton will move back to the initial location ($Holding$). In this location the robot can release the ball again which leads the hybrid automaton to move to the location of $Moving$. It is important to emphasise that the transition from $Moving$ to $Holding$ is optional and it is possible that the hybrid automaton stays in the same state as long as all the invariant conditions are satisfied. Moreover, if the

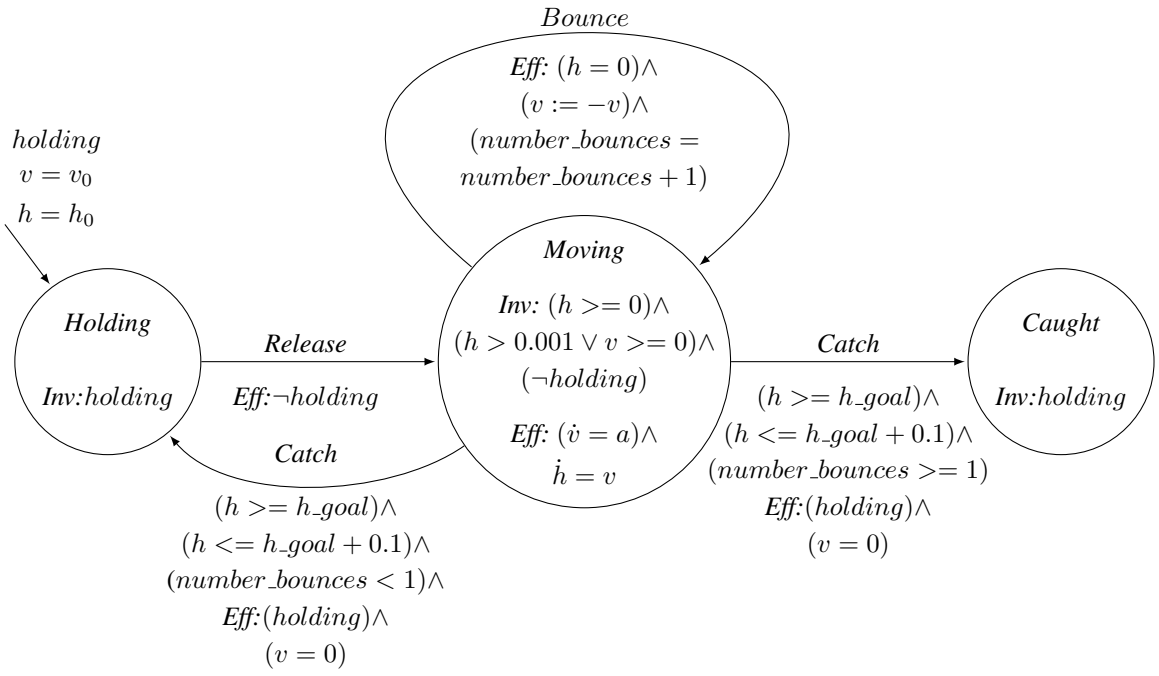


Figure 4: Free Fall hybrid automaton

ball is caught and the number of times that the ball has bounced is at least one, the hybrid automaton will move from the $Moving$ (middle circle) to $Caught$ (right circle) which represents the goal state.

3.3 PDDL+ Model of the Free Fall Problem

In this section we describe how the hybrid automaton of the Free Fall problem can be written in PDDL+. A planning model represented in PDDL+ consists of the domain and the problem instance. The domain model lists the action schema and the possible processes and events. The problem instance describes the objects, the initial state, and the goal state. The domain corresponding to the hybrid automaton in Figure 4 is shown in Figure 5, while the problem instance is shown in Figure 6. In this problem we include a single ball, *ball1*, and define that the robot should catch the ball with $h_{goal} = 5$.

Grounding the problem by applying the objects from the problem instance to the predicates of the domain and also considering the numeric variable in the goal respectively forms the propositional variable $(holding\ ball1)$ and numeric variable $(number\_bounces\ ball1)$. These variables describe a discrete state-space of three states:

- The state in which $(holding\ ball1)$ is false and $(<= (number\_bounces\ ball1)\ 1)$, which is the initial state and also the state reached after the ball is caught by action $Catch$, but the number of bounces is less than one. This corresponds to the $Holding$ (left circle) location of the hybrid automaton.
- The state in which $(holding\ ball1)$ is false. This state corresponds to the $Moving$ (middle circle) location of the hybrid automaton. In this discrete state, the process $(moving\ ball1)$ models the continuous change on the numeric variables.

```
(define (domain dropping_ball)
(:requirements :fluents :typing :durative-actions :time :negative-preconditions
               :duration-inequalities)

(:types ball)
(:predicates (holding ?b - ball))
(:functions
    (velocity ?b - ball)
    (height ?b - ball)
    (h_goal)
    (number_bounces ?b - ball)
    (a)
)

(:action release
 :parameters (?b - ball)
 :precondition (and (holding ?b) (= (velocity ?b) 0))
 :effect (and (not (holding ?b)))
)

(:process moving
 :parameters (?b - ball)
 :precondition (and (not (holding ?b)) (>= (height ?b) 0))
 :effect (and
    (increase (velocity ?b) (* #t (- (a))) )
    (increase (height ?b) (* #t (velocity ?b))))
 )

(:event bounce
 :parameters (?b - ball)
 :precondition (and (< (velocity ?b) 0) (<= (height ?b) 0.001))
 :effect (and
    (increase (number_bounces ?b) 1)
    (assign (velocity ?b) (* -1 (velocity ?b))))
)

(:action catch
 :parameters (?b - ball)
 :precondition (and
    (>= (height ?b) (h_goal))
    (<= (height ?b) (+ (h_goal) 0.1)))
 :effect (and
    (holding ?b)
    (assign (velocity ?b) 0)))
)
```

Figure 5: Simplified PDDL+ domain description for Free Fall.

- The state in which $(holding\ ball1)$ is true and $(>=\ (number\_bounces\ ball1)\ 1)$ is reachable from the $Moving$ state by applying action $catch$ after the ball has bounced at least once. This state corresponds to the $Caught$ (right circle) location in the hybrid automaton. The goal holds in this state.

The continuous change of the location $Moving$ is modelled by the process $(moving\ ball1)$, which is active while its preconditions, $(not\ (holding\ ball1))$ and $(>=\ (height\ ball1)\ 0)$, are satisfied. The event

```
(define (problem dropping_ball_1)
(:domain dropping_ball)
(:objects ball1 - ball)

(:init
    (holding ball1)
    (= (velocity ball1) 0)
    (= (height ball1) 10)
    (= (h_goal) 5)
    (= (number_bounces ball1) 0)
    (= (a) 9.8)
)

(:goal
    (and (holding ball1)
    (>= (number_bounces ball1) 1))
))
```

Figure 6: Simplified PDDL+ Free Fall problem file

$(bounce\ ball1)$ has no effect upon the discrete state, instead affecting only the numeric variables v and $(number\_bounces)$. This event corresponds to the self-looping transition $Bounce$ in the hybrid automaton. As explained previously, an event is triggered as soon as its preconditions are satisfied. In this case, the event will be triggered as soon as the ball has reached the floor. This precondition is shown as $(< (velocity\ ?b)0))$ and $(<=\ (height\ ?b)\ 0.001)$. Note that in hybrid automaton semantics, the discrete transitions follow a *may* semantics: the $Catch$ and $Release$ transitions do not have to be taken as long as the invariant of the current location is not violated. However, in PDDL+, events follow a *must* semantics and are triggered as soon as their preconditions are satisfied. This is modelled in the automaton in the invariant of the $Moving$ location, which forces the $Bounce$ transition to occur when its preconditions are achieved. This translation of *must semantics* between PDDL+ and Hybrid Automata is explained in more detail by Bogomolov et al. (2015).

4. Encoding PDDL+ Domains in SMT

In this section we describe how we encode a PDDL+ domain into SMT. The intuition behind the encoding mirrors that of propositional planning as SAT, described in Section 2.3.

First, a PDDL+ planning problem for a fixed horizon n is modelled by defining n copies of the Boolean and Real variables which describe the state and actions. In contrast to planning as SAT, in which each set of state and action variables referred to a *step*, in PDDL+ each set of variables refers to a time-stamped moment of discrete change, called a *happening*, between which there is only continuous numeric change. The constraints of the SMT formula enforce the discrete change at each happening, and the continuous change between them. A satisfying assignment to the variables corresponds to a plan trace.

First we describe the encoding of the happening, and later we expand the encoding for the whole planning problem.

4.1 Encoding of a single happening

Our encoding is based on the notion of *happening*, which is used to capture the discrete change in the state at a given time point due to the effects of actions, processes, or events. Namely, each happening encodes the causal chain of events, processes and instantaneous actions which might occur simultaneously at a given time point. We have defined a bound B as the length of the causal cascading instantaneous events and we split durative actions as two instantaneous actions, representing the start and end of the action, and one process representing the continuous numeric effects and invariant.

Definition 4 (Happening). A happening is the tuple $x := \{P, V, E, Ps, A\}$, where:

- $P = \{P_0, \ldots, P_{B+1}\}$ are the propositional state variables at happening x, in $B+1$ layers;
- $V = \{V_0, \ldots, V_{B+1}\}$ are the real state variables at happening x, also in $B+1$ layers;
- $E = \{E_0, \ldots, E_B\}$ represents the chain of events triggered at happening x. The chain is of length B;
- Ps represents the set of processes active over the next interval;
- A is the set of actions applied at the happening.

Each P_i is a set of all the propositions, i.e.:

$$P_i = \{p_i, \forall p \in P\}, i \in \{0, \ldots, B+1\}$$

Similarly we have the following for the V_i and E_i:

$$\begin{aligned} V_i &= \{v_i, \forall v \in V\}, \quad i \in \{0, \ldots, B+1\} \\ E_i &= \{e_i, \forall e \in E\}, \quad i \in \{0, \ldots, B\} \end{aligned}$$

An example of happening is shown in Figure 7. The chain of events forms a set of layers at each happening. Each circle in Figure 7 represents a layer. The number of event steps in each happening is bounded to B. The last layer indicates the final value of the state variables after considering the effects of the cascading events. If the causal chain of events is longer than this bound, then a valid plan will not be found.

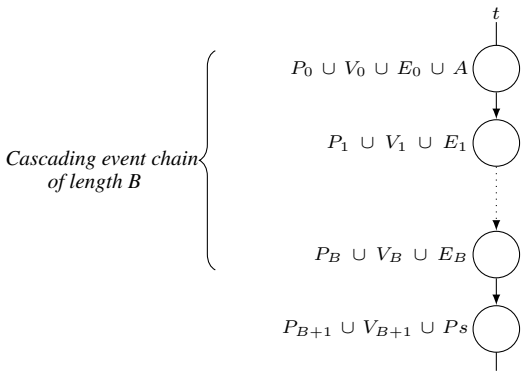


Figure 7: A single happening occurs at time t, and includes several sets of state variables. These sets describe a causal chain of instantaneous events.

Following from Definition 4, happening x_t is encoded by the SMT variables:

$$x_t := \left\langle \begin{array}{l} time_t, \\ P_{0,t}, \ldots, P_{B+1,t}, \\ V_{0,t}, \ldots, V_{B+1,t}, \\ E_{0,t}, \ldots, E_{B,t}, \\ Ps_t, A_t, \\ flow_{V_t}, dur_{Ps_t} \end{array} \right\rangle$$

The happenings in our encoding either describe a moment of discrete change - which corresponds to the discrete transition *Trans* of the hybrid automata - or a point in time between moments of discrete change in which the derivative is equal to zero for some numeric continuous change. The latter case is to ensure that invariant conditions hold, avoiding the case described later in Figure 11. Between happenings there is only

continuous numeric change (*Flow*). The key difference between a hybrid automaton and the SMT encoding is that multiple actions can be performed in a single happening in parallel, meaning that while the hybrid automaton is exponential in the size of the PDDL+ description, our encoding will be linear.

The set $flow_{V_t} := \{flow_{v_t} | \forall v_t \in V_t\}$ is a set of numeric expressions that represent the change in value of each numeric variable v from this time point to the next. The variables $dur_{Ps_t} := \{dur_{ps_t} | \forall ps_t \in Ps_t\}$ represents the remaining duration of each process. dur_{ps} and is constrained to be positive if and only if the process is currently executing.

The constraints within a happening are shown in Figure 8.

Proposition and real variable support

H1. $\bigwedge_{p \in P} p_{1,t} \rightarrow (p_{0,t} \vee \bigvee_{e \in E | p \in eff_e^+} e_{0,t} \vee \bigvee_{a_t \in A | p \in eff_a^+} a_t)$

H2. $\bigwedge_{p \in P} \neg p_{1,t} \rightarrow (\neg p_{0,t} \vee \bigvee_{e \in E | p \in eff_e^-} e_{0,t} \vee \bigvee_{a \in A | p \in eff_a^-} a_t)$

H3. $\bigwedge_{i=1}^{B} \bigwedge_{p \in P} p_{i+1,t} \rightarrow (p_{i,t} \vee \bigvee_{e \in E | p \in eff_e^+} e_{i,t})$

H4. $\bigwedge_{i=1}^{B} \bigwedge_{p \in P} \neg p_{i+1,t} \rightarrow (\neg p_{i,t} \vee \bigvee_{e \in E | p \in eff_e^-} e_{i,t})$

H5. $\bigwedge_{v \in V} (\bigwedge_{a \in A | v \in eff_a^{num}} \neg a_t \wedge \bigwedge_{e \in E | v \in eff_e^{num}} \neg e_{0,t}) \rightarrow (v_{i+1,t} = v_{i,t})$

H6. $\bigwedge_{i=1}^{B} \bigwedge_{v \in V} (\bigwedge_{e \in E | v \in eff_e^{num}} \neg e_{i,t}) \rightarrow (v_{i+1,t} = v_{i,t})$

Event preconditions and effects

H7. $\bigwedge_{i=0}^{B} \bigwedge_{e \in E} e_{i,t} \leftrightarrow pre_e(P_{i,t} \cup V_{i,t})$

H8. $\bigwedge_{i=0}^{B} \bigwedge_{e \in E} e_{i,t} \rightarrow eff_e(P_{i+1,t} \cup V_{i+1,t})$

Action preconditions and effects

H9. $\bigwedge_{a \in A} a_t \rightarrow pre_a(P_{0,t} \cup V_{0,t})$

H10. $\bigwedge_{a \in A} a_t \rightarrow eff_a(P_{1,t} \cup V_{1,t})$

Process triggering

H11. $\bigwedge_{ps \in Ps} ps_t \leftrightarrow pre_{ps}(P_{B+1,t} \cup V_{B+1,t})$

H12. $\bigwedge_{ps \in Ps} dur_{ps_t} >= 0$

H13. $\bigwedge_{ps \in Ps} ps_t \leftrightarrow (dur_{ps_t} > 0)$

Action mutexes

H14. $\bigwedge_{a \in A \cup E} \bigwedge_{a' \in A \cup E | a \# a'} (\neg a_t \vee \neg a'_t)$

Figure 8: Encoding of a PDDL+ happening to SMT.

- Proposition and real variable support

  These constraints ensure that the value of propositions (H1-H4) and real variables (H5-H6) remains consistent from $P_{0,t} \cup V_{0,t}$ to $P_{B+1,t} \cup V_{B+1,t}$. Constraints (H1) and (H2) enforce the correct values after considering the add and delete effects of the initial events ($E_{0,t}$) and actions (A_t). Similarly, constraints (H3) and (H4) enforce the propositional add and delete effects throughout the instantaneous

chain of events. (H5) enforces the numeric effects of the initial events and actions. (H6) enforces the numeric effects of the remaining events. Note that after the first layer at each happening there are only events, and no actions.

- Event preconditions and effects

  This set of constraints enforces that an event is triggered if and only if its precondition holds (H7) and that if an event is triggered, its effects are present in the next layer of that happening (H8).

- Action preconditions and effects

  Similar to (H6-H7), these constraints ensure for actions A_t that their preconditions must hold in $P_{0,t} \cup V_{0,t}$ (H9) and their effects are enforced in $P_{1,t} \cup V_{1,t}$ (H10).

- Process triggering

  This group of constraints enforce that a process is active if and only if its preconditions are satisfied in set $P_{B+1,t} \cup V_{B+1,t}$ (H11). It also enforces that the real variable dur_{ps_t} for each process is greater than or equal to zero (H12), and strictly greater that zero if and only if the process is active at the end of the causal chain of events (H13). These constraints will be used to ensure that a process cannot finish outside of a happening.

- Action mutexes

  This set of binary constraints enforces that no two mutex actions (or a mutex action and event) can be applied simultaneously. For each mutex pair, denoted $a_t \nparallel a'_t$, at least one must be false (H14).

4.2 Encoding of a Plan

Having defined happenings, a PDDL+ problem $\Pi+$ can be encoded as a bounded number of happenings $X := \{x_1, \ldots, x_n\}$, such that any proof for the SMT formula represents the trace of a valid plan for $\Pi+$. The *plan* corresponding to that trace is the set of action assignments $A_1 \cup \ldots \cup A_n$.[3]

Therefore, the existence of a plan for a *PDDL+ planning problem* $\Pi+$ *with bound* n is proved by building the SMT formula $\Pi+_n$ in the theory of quantifier-free (nonlinear) real arithmetic with n copies of the set of happening variables $X = \{x_1, \ldots, x_n\}$ for $n \geq 1$. A *plan for* $\Pi+$ is the assignment to the action variables in any proof of $\Pi+_n$. The encoding is illustrated by Figure 9.

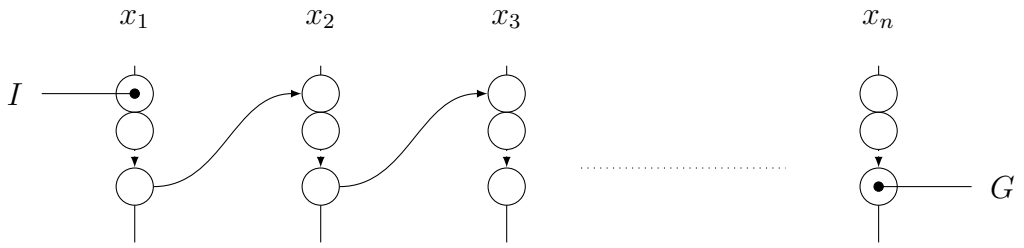


Figure 9: A plan is found by building a formula with n copies of the set of variables x_t for $t = 1 \ldots n$. Each Happening models discrete change. Between happenings there is only continuous numeric change. The initial state is modelled in t_1, and the goal constraints are added to t_n.

The constraints for a happening in Figure 8 are copied for each happening $x_1 \ldots x_n$. Additional constraints in the SMT formula $\Pi+_n$ are shown in Figure 10 and explained below.

- Instance description

3. As processes and events do not appear in a PDDL+ plan.

Instance description

P1. $I(P_{0,1} \cup V_{0,1})$

P2. $G(P_{B+1,n} \cup V_{B+1,n})$

P3. $time_1 = 0$

P4. $\bigwedge_{i=2}^{n} time_i \geq time_{i-1} + \varepsilon$

Proposition support

P5. $\bigwedge_{i=2}^{n} \bigwedge_{p \in P} p_{0,i} \rightarrow p_{B+1,i-1}$

P6. $\bigwedge_{i=2}^{n} \bigwedge_{p \in P} \neg p_{0,i} \rightarrow \neg p_{B+1,i-1}$

Invariants

P7. $\bigwedge_{i=2}^{n} \bigwedge_{ps \in Ps} ps_{i-1} \rightarrow dur_{ps_i} = dur_{ps_{i-1}} - time_i + time_{i-1}$

P8. $\bigwedge_{i=1}^{n-1} \bigwedge_{ps \in Ps} ps_i \leftrightarrow pre_{\leftrightarrow ps_i}$

P9. $\bigwedge_{i=1}^{n-1} \bigwedge_{e \in E} \neg pre_{\leftrightarrow e_i}$

Continuous change on real variables

P10. $\bigwedge_{i=1}^{n-1} \bigwedge_{v \in V} flow_{v_i} = \int_{time_i}^{time_{i+1}} \sum_{ps \in Ps} eff_{\leftrightarrow ps}^{num}(V_i) dt$

P11. $\bigwedge_{i=2}^{n} \bigwedge_{v \in V} (v_{0,i} = v_{B+1,i-1} + flow_{v,i-1})$

Figure 10: Encoding of PDDL+ planning problem $\Pi+$ to SMT.

These constraints enforce the initial state to hold in the first happening (P1), and that the goal is achieved in the final happening (P2). Following PDDL2.1 and PDDL+ semantics, they constrain the timing of happenings to enforce epsilon separation (P3-P4).

- Proposition support

  These constraints ensure that the discrete state variables do not change between happenings (P5-P6).

- Invariants

  These constraints ensure that the continuous numeric change between happenings does not violate any invariant constraints. First (P7) ensures that if a process is active in the previous happening, its duration is decreased by the time between happenings. This constraint, in combination with constraints (H12-H13) of Figure 8, ensure that a process cannot end between happenings. A process can remain active over intervals spanning multiple happenings.

  Constraint (P8) enforces the invariant of the process. If a process is active, then the precondition of the process is active over the whole interval between happenings, and if the process is not active, then its preconditions are false over the whole interval. For constraints over real valued variables, this is done by checking the value either side of the interval. For nonlinear change, this is not sufficient (as shown in Figure 11). It is necessary to include the following additional constraint:

$$ps_{t_j} \rightarrow \bigwedge_{a_{t_j}=1}^{A_{t_j}} \left((\frac{d^a f}{dt^a})_j (\frac{d^a f}{dt^a})_{j+1} >= 0 \right); \tag{1}$$

  where f is the numeric, non-constant part of the invariant, and where the $(A+1)$th derivative of f is identically zero. This ensures that the derivatives of the function do not cross zero over the interval, thus a fluctuating value of f cannot violate the invariant condition between t_j and t_{j+1}. This is explained in more detail in Section 4.3.

  Constraint (P9) similarly ensures that an event is not triggered during an interval.

- Continuous change on real variables

  These constraints enforce continuous change over the interval (P10), applying the result to each real variable (P11). In order to calculate the change, the indefinite integral of the process' effects upon the variable must be computed. This is done automatically using the computer algebra systems (CAS) SymPy (SymPy, 2013) and Piranha (Biscani et al., 2018).

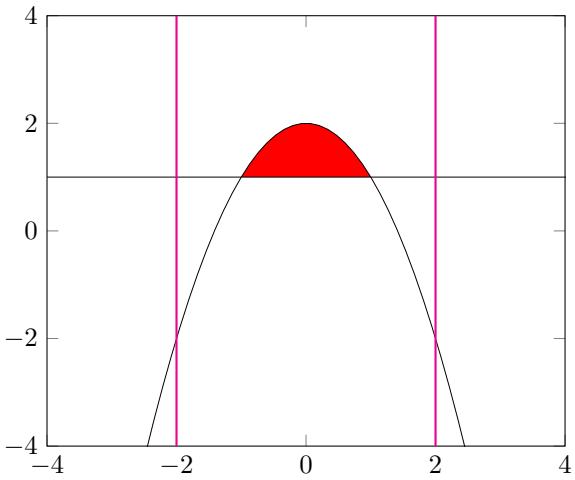


Figure 11: The plot illustrates continuous non-linear numeric change between two time points indicated by vertical lines. While considering the non-linear change, we need to check whether the invariant condition of a process is satisfied throughout the interval. If the invariant asserts that the function should remain below the horizontal line, the figure shows that it is not sufficient to check the value of the function only at the time points.

For example, consider the process of the ball falling in the working example. The ball is accelerating at a constant rate in a single axis. The PDDL process describing the continuous change of the ball's velocity and height is as shown in Figure 12.

```
(:process moving
 :parameters (?b - ball)
 :precondition (and (not (holding ?b)) (>= (height ?b) 0))
 :effect (and
    (increase (velocity ?b) (* #t -9.8) )
    (increase (height ?b) (* #t (velocity ?b))))
 )
```

Figure 12: The process *moving* from the PDDL+ Free Fall domain.

While this process is active, the change in height of the ball during the interval between two happenings is a non-linear function of time. To enable the encoding to describe this change the CAS first produces an expression for the change in height whose terms are time, and constant coefficients. For example:

$$\delta h = \frac{1}{2}at^2 + v_0 t$$

Where v_0 is the initial velocity, and a is the constant acceleration. This is explained in more detail in Section 4.3 and later in Section 5, where the whole encoding of the working example is shown.

Note that in this approach, the integration and differentiation required for P8, P9, and P10 are performed outside the solver, during the encoding. Hence the integration is done only once for each domain.

4.3 Handling Continuous Change

This section provides a more detailed discussion of the constraints governing continuous numeric change and invariants (P8-11). More complete examples are presented in Section 5, which aim to provide a complete view of a single encoding. Instead, here we aim to clarify:

- How to encode multiple processes effecting continuous change on a single real variable.
- The intuition behind the invariant constraint (P8), and how derivatives are used to handle non-linearity.

As an example consider the new process in Figure 13. The process describes the upward force from a thruster on the falling ball. The thruster is only active when switched on, and is automatically cut off if the height of the ball falls below 4.

```
(:process thrusting
 :parameters (?b - ball)
 :precondition (and (not (holding ?b)) (thrusters_on)
                    (>= (height ?b) 4))
 :effect (and (increase (velocity ?b) (* #t 30) )
 )
```

Figure 13: The process *thrusting* added to the PDDL+ Free Fall domain as an example.

Figure 14 shows an example of the continuous change on the height of the ball. At time $t = 0$ the ball is at height 10 and begins to fall. At time $t = 1$ the height of the ball is 5.1 and the thruster is switched on. The fall of the ball is arrested. However, the invariant is violated at time $t = 1.13$ (rounding to 2 decimal places) and the process is no longer active. The ball continues to fall to the ground. The actual height of the ball is shown by the black line in Figure 14.

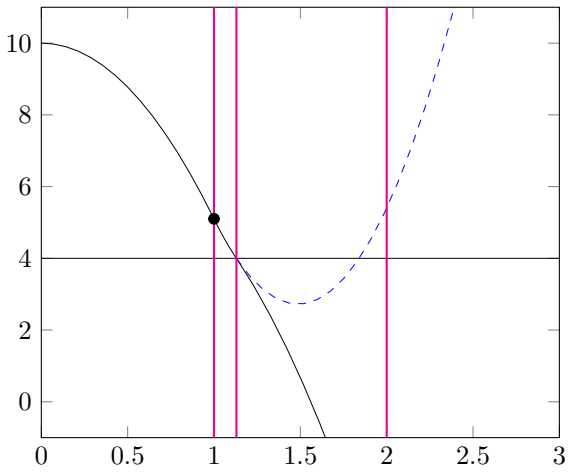


Figure 14: The plot illustrates continuous non-linear numeric change between time points indicated by vertical lines. The thruster is switched on at time $t = 1$, shown by the black point. The invariant of the process is soon violated and the ball will continue to fall. The blue dashed line shows the projected height of the ball if the thruster remains on even after the invariant is violated.

Calculation of Continuous Change

Two processes (*moving* and *thrusting*) affect the velocity of the ball. According to the way in which continuous change accumulates in PDDL+, P10 describes the flow as a linear combination of each active process. In other words, the overall change on the real variable is a sum of the rate of change from each active process. Those rates of change are: -9.8 (from *moving*) and 30 (from *thrusting*).

As integration is a linear function, each process effect can be integrated separately and the result combined to find the total change. Using the *if-the-else* syntax of SMT, the combination of process effects can be

compactly represented. Therefore, $flow_{v_i}$, the total continuous change on the velocity of the ball between $time_i$ and $time_{i+1}$ is:

$$flow_{v_i} = \text{ife}\big(moving_i, \int_{time_i}^{time_{i+1}} -9.8dt, 0\big) + \text{ife}\big(thrusting_i, \int_{time_i}^{time_{i+1}} 30dt, 0\big)$$

$$= \text{ife}\big(moving_i, -9.8(time_{i+1} - time_i), 0\big) + \text{ife}\big(thrusting_i, 30(time_{i+1} - time_i), 0\big) \quad (2)$$

and the final value of the velocity is set by (P11) as:

$$v_{i+1} = v_i + flow_{v_i} \quad (3)$$

The expressions above will be used in the encoding, as the values of the time variables are not known until the problem is solved. However, as an example, consider the height of the ball in Figure 14. The initial velocity $v_0 = 0$. In the first interval between $time_0 = 0$ and $time_1 = 1$ the process *thrusting* is not active, and so the latter *if-then-else* statement takes the value 0. The value of $flow_{v_0}$ is therefore -9.8 and the velocity at $time_1$ is $v_1 = -9.8$.

In the second interval the process *thrusting* is active and so the gradient is increased. The invariant of the process is violated at time 1.13, so the next happening in our example must occur at that time. Substituting into equation (2), the value of $flow_{v_1}$ is:

$$\begin{aligned} flow_{v_1} &= -9.8(time_2 - time_1) + 30(time_2 - time_1) \\ &= -9.8(0.13) + 30(0.13) = 2.626 \end{aligned}$$

and the final velocity at $time_2 = 1.13$ is:

$$v_2 = v_1 + flow_{v_1} = -7.174$$

The height of the ball is calculated in the same way, except that there is only one process that has continuous effect on the height. The expression for the rate of change of h includes v, which is a function of time. Before performing any integration, the height of the ball must first be represented in terms of t and constant coefficients. The CAS is used to perform this substitution:

$$flow_{h_i} = \int_{time_i}^{time_{i+1}} \begin{pmatrix} v_i + \text{ife}(moving_i, -9.8t, 0) \\ +\text{ife}(thrusting_i, 30t, 0) \end{pmatrix} dt \quad (4)$$

As before, each component can be integrated separately to produce the final values for the change in h and the value of h at the end of the interval:

$$flow_{h_i} = \Big(v_i(time_{i+1} - time_i)$$

$$+\text{ife}(moving_i, -4.9(time_{i+1} - time_i)^2, 0) + \text{ife}(thrusting_i, 15(time_{i+1} - time_i)^2, 0) \Big)$$

$$h_{i+1} = h_i + flow_{h_i}$$

EXAMPLE OF INVARIANTS AND ZERO-CROSSING

To ensure that the invariant constraint on linear continuous change is not violated, it is sufficient to check the value of the function at the boundaries of the interval. However, with non-linear change this is not sufficient. The constraints at the boundaries are not sufficient as the non-linear change may contain a turning point that allows the value to violate the invariant while remaining valid at the boundaries, as explained above with Figure 11.

Constraint (P8) approaches this problem by including that every derivative of the function cannot cross zero within an interval. The intuition behind this approach is that a turning point (necessary for the zero-crossing problem to occur) cannot happen if the derivative of the function does not cross zero. The function must be monotonic within the interval. Therefore, it is guaranteed that the invariant condition is satisfied.

As an example, the constraint on the *thrusting* process is $h - 4 \geq 0$. Enforcing this constraint at each happening (the interval boundaries) produces the following constraint:

$$\bigwedge_{i=1}^{n-1} h_i - 4 \geq 0$$

In Figure 14 the height of the ball falls below 4 at time 1.13. This violates the invariant condition on the process *thrusting*. However, if the next happening occurs at time $t = 2$ then the boundary constraint above would still be satisfied. The height of the ball in this erroneous case is illustrated in Figure 14 with a blue dashed line.

The derivative of h is computed using the CAS and equation (4).

$$\frac{\delta flow_{h_i}}{\delta t} = v_i + \text{ife}(moving_i, -9.8t, 0) + \text{ife}(thrusting_i, 30t, 0) \tag{5}$$

Note that the derivative of the height is the velocity. When this rate of change is exactly zero, there is a turning or inflection point in the function. This derivative is plotted in Figure 15. Note that the derivative crosses zero exactly at the turning point, at which time the invariant is violated.

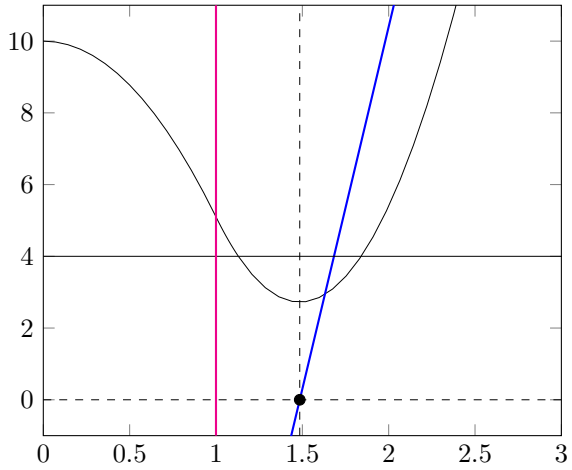


Figure 15: The plot illustrates continuous non-linear numeric change between time points indicated by vertical lines. The derivative of the function is shown by the blue line.

Following the intuition described above, it is not allowed for the rate of change to cross zero within an interval. Therefore, the constraint (P8) includes that the derivative cannot cross zero, as shown in equation (1) in Section 4.2, and using the derivative of h from equation (5):

$$\bigwedge_{i=1}^{n-1} \left(\frac{\delta flow_{h_{i+1}}}{\delta t}\right)\left(\frac{\delta flow_{h_i}}{\delta t}\right) \geq 0$$

This ensures that the value of h is monotonically increasing or decreasing within each interval. In the example of Figures 14 and 15, the next happening can come no later than the point at which the derivative crosses zero. As the invariant is violated at this happening, the constraints forbid the erroneous behaviour that allow the thruster to remain on.

Note that the derivative may also be a non-linear function, and might also contain a turning point. Therefore, as shown in equation (1) all of the non-constant derivatives must be included. This ensures that the function does not contain a turning point, and therefore by checking the value of the function at the boundaries of the interval, it is ensured that an invariant is not violated. It is for this reason that the approach is limited to those non-linear functions that have an $(A + 1)$th derivative that is identically zero. Invariants on functions that fluctuate infinitely often, such as sin waves, cannot be ensured with this approach.

4.4 SMTPlan Architecture

The encoding has been implemented in a planner, called SMTPlan. The planner follows the procedure of SATPlan, iteratively deepening on the number of happenings (as opposed to the number of steps). The number of happenings can be incremented by step size s, which is an integer parameter that the user can specify. Given a planning problem $\Pi+$, the algorithm of SMTPlan is as follows:

1. Perform the indefinite integration and differentiation on the domain of $\Pi+$.
2. Encode $\Pi+$ with an initial bound of n happenings, as $\Pi+_n$.
3. Pass $\Pi+_n$ to SMT solver to attempt to find a satisfying assignment.
4. If $\Pi+_n$ is unsatisfiable, then n is incremented ($n = n + s$) and the problem is encoded and passed to the solver again.
5. If satisfiable, then extract the truth assignment to the action variables $A_0, \ldots, A_n$ and return.

The algorithm is shown in Algorithm 1 and the steps described in more detail below. The process is illustrated in Figure 16. Note that while it is technically possible to iterate on the bound of the event cascade, it is not the behaviour of our implementation. The number of events that occur in a chain in most domains is typically very low, and the default bound is sufficient to capture the whole chain. Moreover, for domains in which the event chain is longer, it is possible to set an initial bound that is high enough and iterate only on the number of happenings.

Algorithm 1: SMTPlan

```
1  Input: Planning Problem Π, initial bound n, step size s.
2  ξ ← extractExpressions(Π);            // Extract expressions from domain of Π.
3  Loop
4  |  Π_n ← encode(Π,ξ,n);                  // Extend encoding to n happenings.
5  |  M ← solve(Π_n);                        // Find satisfying assignment.
6  |  if M = ⊥ then
7  |  |  n = n + s;                          // Increment the happening bound.
8  |  else
9  |  |  π ← extractPlan(M);   // Extract a plan from the satisfying assignment.
10 |  |  return;
11 |  end
12 EndLoop
```

The indefinite integration and differentiation required for constraints enforcing the checking of invariant conditions (P8-P9) and continuous numeric change (P10-P11) is performed by extracting the expressions from the PDDL+ domain of $\Pi+$ and passing to the CAS (line 2). In the current implementation of SMTPlan the CAS piranha is used, as it is able to solve polynomial non-linear integrals. The resulting expressions are used in all subsequent encoding steps.

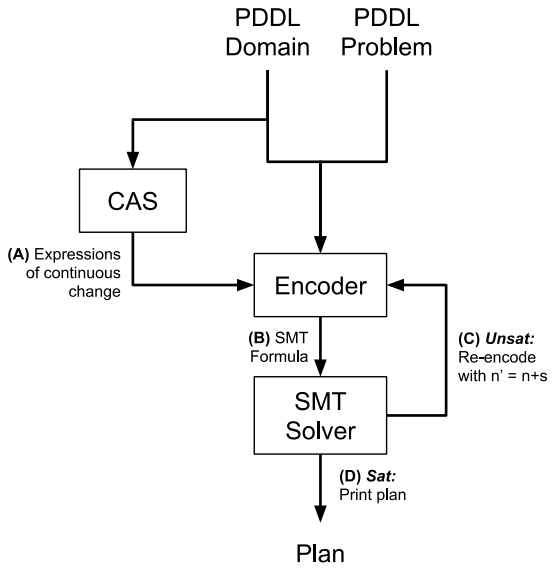


Figure 16: An overview of the SMTPlan architecture. (A) First the Computer Algebra System performs the indefinite integration and differentiation to produce the expressions of continuous numeric change. (B) The PDDL problem instance is encoded as a SMT formula, with an initial number of happenings $n = 2$. The encoding is passed to an SMT solver. (C) If the encoding is unsatisfiable, then the number of happenings is incremented. (D) Finally, if the encoding is satisfiable, the plan is printed.

The implementation of the encoding step (line 4) uses the c++ interface of the SMT-solver z3. This interface allows the planner to incrementally add constraints and call the solver. The benefit of this incremental solving is that each happening only needs to be encoded once, and the iterations are encoded in time linear to the size of the happening and independent of the number of happenings. For example, if an encoding with 4 happenings was found to be unsatisfiable, then extending the encoding to 5 happenings only requires adding the variables and constraints of happening x_5 to the $z3$ interface.

Extracting the plan from the satisfying assignment (line 9) is performed by returning all the variables $a \in A_i$ for each $i = 1 \ldots n$ that are assigned *true*. This corresponds with the actions applied in each happening in order to achieve the goal. Note that the algorithm will return a plan with the minimum number of happenings, which is not necessarily indicative of plan quality. For example, consider a travelling problem in which the solution of best quality is the shortest in time, and achieved by performing many acceleration actions. The first solution found by this algorithm will instead consist of only a single acceleration action and a much longer duration.

5. Example Encodings

In this section we describe the encoding of three different problems in SMT. The first one will be a simplified version of Generator domain in order to show the encoding of continuous change. Then we consider the full

Generator domain. Finally, we consider the Free Fall problem which has been introduced as our working example.

5.1 Simplified Generator

In this section for the sake of showing the encoding of the continuous changes, we use a simplified version of the Generator domain. First we describe the encoding of a PDDL+ benchmark, the *simplified generator* domain. The domain and the problem files are shown in Figures 17 and 18, respectively. The initial state asserts that there is one generator; the generator's capacity is 1060; and initial fuel level is 1020. The goal state is that the generator has been run for a given amount of time: `(generator-ran)`.

We will show the encoding of two happenings x_1 and x_2, the minimum number of steps required to solve this problem. As there are no events in the domain, we will impose a bound $B = 0$ on the number of cascading events. Therefore each happening is the set:

$$x_t := \left\langle \begin{array}{ll} time_t & : Real \\ gen\_ran_{0,t}, gen\_ran_{1,t} & : Bool \\ fuelLevel_{0,t}, fuelLevel_{1,t}, capacity_{0,t}, capacity_{1,t} & : Real \\ generate_{sta,t}, generate_{end,t} & : Bool \\ generate_t & : Bool \\ generate_{dur,t} & : Real \end{array} \right\rangle$$

```
(define (domain simple_generator)
(:requirements
    :fluents :durative-actions
    :duration-inequalities :adl
    :typing)

(:types generator)

(:predicates
    (generator-ran)

(:functions
    (fuelLevel ?g - generator)
    (capacity ?g - generator))

(:durative-action generate
 :parameters (?g - generator)
 :duration (= ?duration 1000)
 :condition (over all (>= (fuelLevel ?g) 0))
 :effect (and
    (decrease (fuelLevel ?g) (* #t 1))
    (at end (generator-ran)))))
```

Figure 17: Simplified PDDL+ generator domain file

```
(define (problem simple_generator)
(:domain simple_generator)
(: objects gen - generator)
(:initial
    (= (fuelLevel gen) 1020)
    (= (capacity gen) 1060))

(:goal
    (generator-ran)))
```

Figure 18: Simplified PDDL+ generator problem file

Proposition support constraints (within happenings) and action preconditions and effects, describe the discrete changes that occur within a happening (H1-H10):

$$
\begin{aligned}
&gen\_ran_{1,1} \rightarrow (gen\_ran_{0,1} \vee generate_{end,1}) \\
&\neg gen\_ran_{1,1} \rightarrow \neg gen\_ran_{0,1} \\
&gen\_ran_{1,2} \rightarrow (gen\_ran_{0,2} \vee generate_{end,2}) \\
&\neg gen\_ran_{1,2} \rightarrow \neg gen\_ran_{0,2} \\
&\\
&fuelLevel_{0,1} = fuelLevel_{1,1} \\
&fuelLevel_{0,2} = fuelLevel_{1,2} \\
&\\
&generate_{sta,1} \rightarrow (fuelLevel_{0,1} >= 0) \\
&generate_{sta,2} \rightarrow (fuelLevel_{0,2} >= 0) \\
&generate_{end,1} \rightarrow (fuelLevel_{0,1} >= 0) \\
&generate_{end,2} \rightarrow (fuelLevel_{0,2} >= 0) \\
&\\
&generate_{sta,1} \rightarrow (generate_{dur,1} = 1000.0) \\
&generate_{sta,2} \rightarrow (generate_{dur,2} = 1000.0) \\
&\\
&\neg generate_{end,1} \\
&generate_{end,2} \rightarrow generate_1 \\
&generate_{end,2} \rightarrow (generate_{dur,2} = 0.0) \\
&generate_{end,1} \rightarrow (generate_{dur,1} = 0.0) \\
&generate_{end,2} \rightarrow gen\_ran_{1,2} \\
&generate_{end,1} \rightarrow gen\_ran_{1,1}
\end{aligned}
$$

Process triggering constraints work together to ensure that the durative actions begin and end within a happening (H11-H13):

$$
\begin{aligned}
&generate_1 = (generate_{dur,1} > 0) \\
&\neg generate_1 = generate_{dur,1} = 0 \\
&generate_2 = (generate_{dur,2} > 0) \\
&\neg generate_2 = generate_{dur,2} = 0
\end{aligned}
$$

Finally, action mutexes are included (H14). In our encoding, we make the starts and ends of durative actions mutually exclusive, for $i = \{1,2\}$:

$$\neg gen\_start_i \vee \neg gen\_end_i$$

The instance description enforces the initial state, the goal condition, and that the second happening is at least ε after the first (P1-P4).

$$\neg gen\_ran_{0,1}$$
$$fuelLevel_{0,1} = 1020$$
$$capacity_{0,1} = 1060$$
$$gen\_ran_{1,2}$$
$$time_1 = 0$$
$$time_2 >= time_1 + 0.001$$

Proposition support constraints (between happenings) ensure that the discrete state stays constant during the interval between happenings (P5-P6):

$$gen\_ran_{1,1} = gen\_ran_{0,2}$$

Invariant constraints ensure that the durative action's overall condition holds over an interval in which its associated process is active, and that the durative action's duration is properly updated across the interval (P7-P9):

$$generate_1 \rightarrow (fuelLevel_{0,1} >= 0)$$
$$generate_2 \rightarrow (fuelLevel_{0,2} >= 0)$$
$$generate_1 \rightarrow (generate_{dur,2} = generate_{dur,1} + time_1 - time_2)$$

The continuous change over real variables is defined and enforced by the $flow$ variables (P10-P11):

$$generate_1 \rightarrow fuelLevel_{0,2} = fuelLevel_{1,1} - time_2 + time_1$$
$$\neg generate_1 \rightarrow (fuelLevel_{0,2} = fuelLevel_{1,1})$$
$$capacity_{0,2} = capacity_{0,1}$$

The resulting plan is shown by assignment to variables in Table 1. The plan corresponding to the assignment presented in Table 1 is shown in Figure 19.

x_1	x_2
$t_1 := 0$	$t_2 := 1000$
$gen\_ran_{0,1} := 0$ $gen\_ran_{1,1} := 0$	$gen\_ran_{0,2} := 1$ $gen\_ran_{1,2} := 1$
$fuelLevel_{0,1} := 1020.0$ $fuelLevel_{1,1} := 1020.0$ $capacity_{0,1} := 1060.0$ $capacity_{1,1} := 1060.0$	$fuelLevel_{0,2} := 20.0$ $fuelLevel_{1,2} := 20.0$ $capacity_{0,2} := 1060.0$ $capacity_{1,2} := 1060.0$
$generate_1 := 1$	$generate_2 := 0$
$\mathbf{generate_{sta,1} := 1}$ $generate_{end,1} := 0$	$generate_{sta,2} := 0$ $\mathbf{generate_{end,2} := 1}$
$generate_{dur,1} := 1000.0$	$generate_{dur,2} := 0$

Table 1: The SMT variables and the assignments found by SMTPlan for the simple generator domain, as assignment to the variables. Boolean variables that are true are assigned as 1 and the ones that are false are 0.

```
0.0: generate [1000.0]
```

Figure 19: Plan for the simplified generator problem.

5.2 Generator

In this section we consider the original version of the Generator domain which is shown in Figure 20. Most of the assumptions and conditions of the domain and the problem file remain the same. However, we added a new action *refuel* in the domain which uses a tank to refuel the generator. By applying this action, the amount of the fuel continuously increases by two units per time unit.

```
(define (domain generator_linear)
(:requirements :fluents :durative-actions :duration-inequalities :adl :typing)
(:types generator tank)
(:predicates (refuelling ?g - generator) (generator-ran) (available ?t - tank))

(:functions (fuelLevel ?g - generator) (capacity ?g - generator) )

(:durative-action generate
 :parameters (?g - generator)
 :duration (= ?duration  1000)
 :condition (over all (>= (fuelLevel ?g) 0))
 :effect (and (decrease (fuelLevel ?g) (* #t 1))
  (at end (generator-ran))))

(:durative-action refuel
 :parameters (?g - generator ?t - tank)
 :duration  (= ?duration  10)
 :condition (and (at start (available ?t))
  (over all (< (fuelLevel ?g) (capacity ?g))))
  :effect (and (at start (refuelling ?g))
              (increase (fuelLevel ?g) (* #t 2))
  (at start (not (available ?t)))
  (at end (not (refuelling ?g)))
         )
))
```

Figure 20: Original PDDL+ generator domain file

The problem instance that we have considered is shown in Figure 21. The initial fuel is 990 units and we have just one tank available in our problem.

```
(define (problem run-generator2)
    (:domain generator_linear)
    (:objects gen - generator tank1 - tank)
    (:init
(= (fuelLevel gen)  990)
(= (capacity gen)  1000)
(available tank1)
    )
(:goal (generator-ran))
)
```

Figure 21: Original PDDL+ generator problem file

Table 2 shows the plan for the Generator domain with the assigned variables found by SMTPlan. The plan corresponding to these assignments is shown in Figure 22. In the first column of the table, happening $x1$ corresponds to the initial state, at which time the generate action is started. Thus, the variable $generate_{sta,1}$

is assigned true. The second column corresponds to the beginning of the refuel action, which can also be seen in the positive assignment to variable $refuel\_gen\_tank1_{sta,2}$. In the third column, representing the happening at time 1000, both actions end.

x_1	x_2	x_3
$t_1 := 0$	$t_2 := 990$	$t_3 := 1000$
$gen\_ran_{0,1} := 0$ $gen\_ran_{1,1} := 0$ $available\_tank1_{0,1} := 1$ $available\_tank1_{1,1} := 1$ $refuelling_{0,1} := 0$ $refuelling_{1,1} := 0$	$gen\_ran_{0,2} := 1$ $gen\_ran_{1,2} := 1$ $available\_tank1_{0,2} := 1$ $available\_tank1_{1,2} := 0$ $refuelling_{0,2} := 0$ $refuelling_{1,2} := 1$	$gen\_ran_{0,3} := 1$ $gen\_ran_{1,3} := 1$ $available\_tank1_{0,3} := 0$ $available\_tank1_{1,3} := 0$ $refuelling_{0,3} := 0$ $refuelling_{1,3} := 0$
$fuelLevel_{0,1} := 990.0$ $fuelLevel_{1,1} := 990.0$ $capacity_{0,1} := 1000.0$ $capacity_{1,1} := 1000.0$	$fuelLevel_{0,2} := 0.0$ $fuelLevel_{1,2} := 0.0$ $capacity_{0,2} := 1000.0$ $capacity_{1,2} := 1000.0$	$fuelLevel_{0,3} := 10.0$ $fuelLevel_{1,3} := 10.0$ $capacity_{0,3} := 1000.0$ $capacity_{1,3} := 1000.0$
$generate_1 := 1$ $refuel\_gen\_tank1_1 := 0$	$generate_2 := 1$ $refuel\_gen\_tank1_2 := 1$	$generate_3 := 0$ $refuel\_gen\_tank1_3 := 0$
$\mathbf{generate_{sta,1} := 1}$ $generate_{end,1} := 0$ $refuel\_gen\_tank1_{sta,1} := 0$ $refuel\_gen\_tank1_{end,1} := 0$	$generate_{sta,2} := 0$ $generate_{end,2} := 0$ $\mathbf{refuel\_gen\_tank1_{sta,2} := 1}$ $refuel\_gen\_tank1_{end,2} := 0$	$generate_{sta,3} := 0$ $\mathbf{generate_{end,3} := 1}$ $refuel\_gen\_tank1_{sta,3} := 0$ $\mathbf{refuel\_gen\_tank1_{end,3} := 1}$
$generate_{dur,1} := 1000.0$ $refuelling_{dur,1} := 10.0$	$generate_{dur,2} := 10.0$ $refuelling_{dur,2} := 10.0$	$generate_{dur,3} := 0.0$ $refuelling_{dur,3} := 0.0$

Table 2: The SMT variables and the assignments found by SMTPlan for the original generator domain, as assignment to the variables. Boolean variables that are true are assigned as 1 and the ones that are false are 0.

```
0.0  : generate [1000.0]
990.0: refuel_gen_tank1 [10.0]
```

Figure 22: Plan for the original generator problem.

The continuous change on the numeric variable *fuelLevel* is modelled as described in Section 4.3. There are two processes that can affect the variable: *refuel* and *generate*. The linear combination of these effects are represented as:

$$flow_{fuellevel_i} = \text{ife}\big(refuel\_gen\_tank1_i, \textstyle\int_{time_i}^{time_{i+1}} 2dt, 0\big) + \text{ife}\big(generate_i, \textstyle\int_{time_i}^{time_{i+1}} -1dt, 0\big)$$

After integration and encoded in the SMT formula, the continuous change between the final two happenings appears as:

$$\begin{aligned} fuelLevel_{0,3} = \quad & fuelLevel_{1,2} \\ & + \text{ife}\big(refuel\_gen\_tank1_i, 2 * (t_3 - t_2), 0\big) \\ & + \text{ife}\big(generate_2, t_2 - t_3, 0\big) \end{aligned}$$

From Table 2 it can be seen that both processes are running ($generate_2$ and $refuel\_gen\_tank1_2$ are both positive). The fuel level at happening x_2 is 0 and the timestamps of happenings x_2 and x_3 are 990 and 1000 respectively. Assigning these values into the clause above results in the continuous change:

$$fuelLevel_{0,3} = 0 + 20 - 10$$

5.3 Free Fall

We consider the working example introduced earlier: *Free Fall* problem. We release the ball from the height of 10 meters from the ground (initial velocity is zero). The goal is to catch the ball after it has bounced at least once and the height of it is between 5 and 5.01 meters. The domain and problem files are shown in Figures 5 and 6, respectively.

In order to describe the SMT encoding, first we define the SMT variables used in our encoding, for a single happening x_t. Since the number of events in our domain is equal to one, the bound of cascading events in our problem is fixed as one, so we just have $E_{0,t}$. This means the number of internal layers at each happening is equal to 2. Below we show the constraints on one happening (x_1), then the constraints describing the transition relation in an encoding with three happenings $x_1, \ldots, x_3$.

$$x_t := \left\langle \begin{array}{ll} time_t & : Real \\ holding_{0,t}, holding_{1,t} & : Bool \\ height_{0,t}, height_{1,t}, velocity_{0,t}, velocity_{1,t}, number\_bounces_{0,t}, number\_bounces_{1,t} & : Real \\ bounce_{0,t} & : Bool \\ moving_t & : Bool \\ release_t, catch_t & : Bool \\ moving_{dur,t} & : Real \end{array} \right\rangle$$

As we can see in section 4.1, the first group of constraints are related to *proposition and real variable support*. The following constraints show $H1$ and $H2$:

$$holding_{1,1} \rightarrow (holding_{0,1} \vee catch_1)$$
$$\neg holding_{1,1} \rightarrow (\neg holding_{0,1} \vee release_1)$$

Constraint $H5$ is modelled as following:

$$(\neg bounce_{0,1} \wedge \neg catch_1) \rightarrow (velocity_{1,1} = velocity_{0,1})$$
$$\neg bounce_{0,1} \rightarrow (number\_bounces_{1,1} = number\_bounces_{0,1})$$
$$height_{0,1} = height_{1,1}$$

Since the bound on the number of cascading events is 1, there are no constraints required for $H3$,$H4$ and $H6$. The next group of constraints are *event preconditions and effects*, which are encoded as following (corresponding to $H7$ and $H8$):

$$bounce_{0,1} = ((height_{1,0} <= 1/1000) \wedge (velocity_{1,0} < 0))$$
$$bounce_{0,1} \rightarrow (velocity_{1,1} = -velocity_{1,0})$$
$$bounce_{0,1} \rightarrow (number\_bounces_{1,1} = number\_bounces_{0,1} + 1)$$

Similar to the events, the next group of constraints are *actions preconditions and effects* which correspond to $H9$ and $H10$.

$$release_1 \rightarrow holding_{0,1}$$
$$release_1 \rightarrow (velocity_{0,1} = 0)$$
$$catch_1 \rightarrow (height_{0,1} >= 5)$$
$$catch_1 \rightarrow (height_{0,1} <= 5.01)$$
$$catch_1 \rightarrow (number\_bounces_{0,1} >= 1)$$
$$release_1 \rightarrow \neg holding_{1,1}$$
$$catch_1 \rightarrow holding_{1,1}$$
$$catch_1 \rightarrow (velocity_{1,1} = 0)$$

The constraints $H11$, $H12$, and $H13$ related to the *process triggering* are encoded as follows:

$$moving_1 = \neg holding_{1,1} \wedge (height_{1,1} >= 0)$$
$$moving_1 = (moving_{dur,1} > 0)$$
$$moving_{dur,1} >= 0$$

Since the two actions that we have are not mutex, the constraint related to that ($H14$) is not needed. The next series of constraints encode the transition of one happening to the next happening. We start with *Instance description* constraints of the problem ($P1 - P4$). First the initial state of the problem (corresponding to $P1$):

$$holding_{0,1}$$
$$velocity_{0,1} = 0$$
$$height_{0,1} = 10$$
$$number\_bounces_{0,1} = 0$$

The goal state (corresponding to $P2$):

$$holding_{1,3} number\_bounces_{1,3} >= 1$$

The timings of the happenings (corresponding to $P3$ and $P4$):

$$time_1 = 0$$
$$time_2 >= time_1 + 0.001$$
$$time_3 >= time_2 + 0.001$$

The next group of constraints is *Proposition support* (corresponding to $P5$ and $P6$). Here we show only the transition constraints between happenings $x1$ and $x2$, which are then duplicated for the transition between $x2$ and $x3$.

$$holding_{0,2} \rightarrow holding_{1,1}$$
$$\neg holding_{0,2} \rightarrow \neg holding_{1,1}$$

Invariants are the next group of constraints (corresponding to $P7 - P9$). The derivatives of the numeric variables affected by the continuous process *moving* are used in these constraints to ensure that no interval between two happenings includes a turning point in their function.

$$moving_1 \rightarrow (moving_{dur,2} = moving_{dur,1} + time_1 - time_2)$$
$$moving_1 \rightarrow (height_{0,2} * height_{1,1} >= 0)$$
$$moving_1 \rightarrow (velocity_{0,2} * velocity_{1,1} >= 0)$$

The last group of constraints ensures the *continuous change on the real variables*. The kinematic equations of the free fall problem are the following:

$$v = v_0 - 9.8t$$
$$h = 1/2at^2 + v_0 t + h_0$$

These equations are read directly from the PDDL domain in Figure 5. The effects are passed to the CAS and simplified so that they are expressed in terms of t and constant coefficients. Based on these equations, the change in height and the velocity of the ball can be encoded in constraints as follows:

$$moving_1 \rightarrow (velocity_{0,2} = -9.8 * (time_2 - time_1) + velocity_{1,1})$$
$$moving_1 \rightarrow (height_{0,2} = -4.9 * (time_2 - time_1)^2 + velocity_{1,1} * (time_2 - time_1) + height_{1,1})$$
$$\neg moving_1 \rightarrow (velocity_{0,2} = velocity_{1,1})$$
$$\neg moving_1 \rightarrow (height_{0,2} = height_{1,1})$$

These constraints ensure that if the process is active, then the change in height and velocity of the ball is as described by the kinematic equations. Further, if the process is not active then the velocity and height of the ball remains unchanged. As there is only one continuous process in the domain with an effect including the height and velocity of the ball, these constraints are sufficient to encode the continuous change on those variables. Section 4.3 provides a more complex example in which more than one process acts upon the same variable.

```
0.0      : release      [0.0]
1.850374 : catch ball1  [0.0]
```

Figure 23: Plan for the Free Fall problem.

x_1	x_2	x_3
$time_1 := 0$	$time_2 := 1.4285$	$time_3 := 1.850374$
$holding_{0,1} := 1$ $holding_{1,1} := 0$	$holding_{0,2} := 0$ $holding_{1,2} := 0$	$holding_{0,3} := 0$ $holding_{1,3} := 1$
$height_{0,1} := 10$ $height_{1,1} := 10$ $velocity_{0,1} := 0$ $velocity_{1,1} := 0$ $number\_bounces_{0,1} := 0$ $number\_bounces_{1,1} := 0$	$height_{0,2} := 0.001$ $height_{1,2} := 0.001$ $velocity_{0,2} := -13.9993$ $velocity_{1,2} := 13.9993$ $number\_bounces_{0,2} := 0$ $number\_bounces_{1,2} := 1$	$height_{0,3} := 5.03486$ $height_{1,3} := 5.03486$ $velocity_{0,3} := 9.864924$ $velocity_{1,3} := 0$ $number\_bounces_{0,3} := 1$ $number\_bounces_{1,3} := 1$
$bounce_{0,1} := 0$	$bounce_{0,2} := 1$	$bounce_{0,3} := 0$
$moving_1 := 1$	$moving_2 := 1$	$moving_3 := 0$
$\mathbf{release_1 := 1}$ $catch_1 := 0$	$release_2 := 0$ $catch_2 := 0$	$release_3 := 0$ $\mathbf{catch_3 := 1}$
$moving_{dur,1} := 1.850374$	$moving_{dur,2} := 0.421874$	$moving_{dur,3} := 0$

Table 3: A plan (the assignment to the SMT variables) for the free fall problem domain. True assignments to action variables are shown in bold.

When the above encoding is passed to an SMT solver, such as Z3, the solver attempts to find an assignment of values to the variables that will satisfy every constraint. Such a satisfying assignment corresponds to a valid plan trace. Table 3 shows the SMT variables and the values assigned to them, corresponding to a valid plan trace. The corresponding plan generated by selecting each action variable assign the value $true$, is shown in Figure 23.

At the end, the plan found by SMTPlan is validated using VAL (Howey et al., 2004). The following shows the report generated by VAL and the updating process of the numeric and propositional variables. This part is representing the first column of Table 3 in which the ball is released from height 10. Note that VAL uses phrase *Event triggered* when a process is started and this is different from the events in PDDL+. So at time 0, the process of moving is started and there is no events triggered at this time point.

Time **Details**

0: Deleting (holding ball1)

0: **Event triggered!**
Activated process (moving ball1)

At time point 1.4285, the ball reaches the ground and the precondition of the event bouncing becomes true and it triggers this event. In the second column of Table 3, the effects of this event negate the velocity of the ball. Also, the variable $moving_{dur,2}$ is calculated as the result of deducting the time that has passed since x_1.

Time	Details
1.4285:	**(velocity ball1)**$(t) = -9.8t$ **(height ball1)**$(t) = -4.9t^2 + 10$ Updating **(velocity ball1)** (0) by -13.9993 for continuous update. Updating **(height ball1)** (10) by 0.001 for continuous update.
1.4285:	**Event triggered!** *Triggered event* (bounce ball1) Increasing **(number_bounces ball1)** (0) by 1. Updating **(velocity ball1)** (-13.9993) by 13.9993 assignment.

Finally at time 1.85037, the planner suggests to catch the ball. At this time point the height of the ball is 5.03486 meters (as shown in the third column of Table 3), however the (h_{goal}) defined in the problem file is 5 meters. The reason that the planner accepts this height as the (h_{goal}) can be explained based on the tolerance that has been defined in the domain. This tolerance is shown as $(<= (height?b)(+(h_{goal})0.1))$ in the preconditions of the $catch$ action which gives a tolerance of 0.1 to the (h_{goal}).

Time	Details
1.85037:	**(velocity ball1)**$(t) = -9.8t + 13.9993$ **(height ball1)**$(t) = -4.9t^2 + 13.9993t + 0.001$ Updating **(velocity ball1)** (13.9993) by 9.86492 for continuous update. Updating **(height ball1)** (0.001) by 5.03486 for continuous update.
1.85037:	Adding (holding ball1) Updating **(velocity ball1)** (9.86492) by 0 assignment.
1.85037:	**Event triggered!** *Unactivated process* (moving ball1)

Also, Figures 24 and 25 show the changes of the height and the velocity of the ball during the execution of the plan. The velocity of the ball is negative until the ball bounces at 1.4285 seconds after releasing the ball.

6. Planning with Control Parameters as SMT

PDDL+ is known as an expressive modelling language, however it enforces a significant limitation on the modelling of actions, restricting the action parameters to choose their values from finite domains. The parameter domains for actions are specified in the problem description, enumerated as a list of objects. Using this enumeration, the action schemes can be grounded, producing a finite set of grounded actions. The only exception to finite domain parameters, introduced in PDDL2.1, is the duration of durative actions. The duration of a durative action can be chosen by the planner. This makes possible the modelling of duration dependent effects. Without the flexible duration parameter, the state space is always locally finite, which is to say that only finitely many states are reachable from a given initial state, using plans bounded by a given finite length. The addition of numeric state variables to the classical propositional language allows for the construction of an infinite state space (Savas et al., 2016).

Savas et al. (2016) introduces new set of numeric action parameters called *control parameters*: a generalised version of the duration parameter, which allows the planner to choose their values from an infinite domain. The control parameters can appear both in the conditions and the effects of a durative action. Control

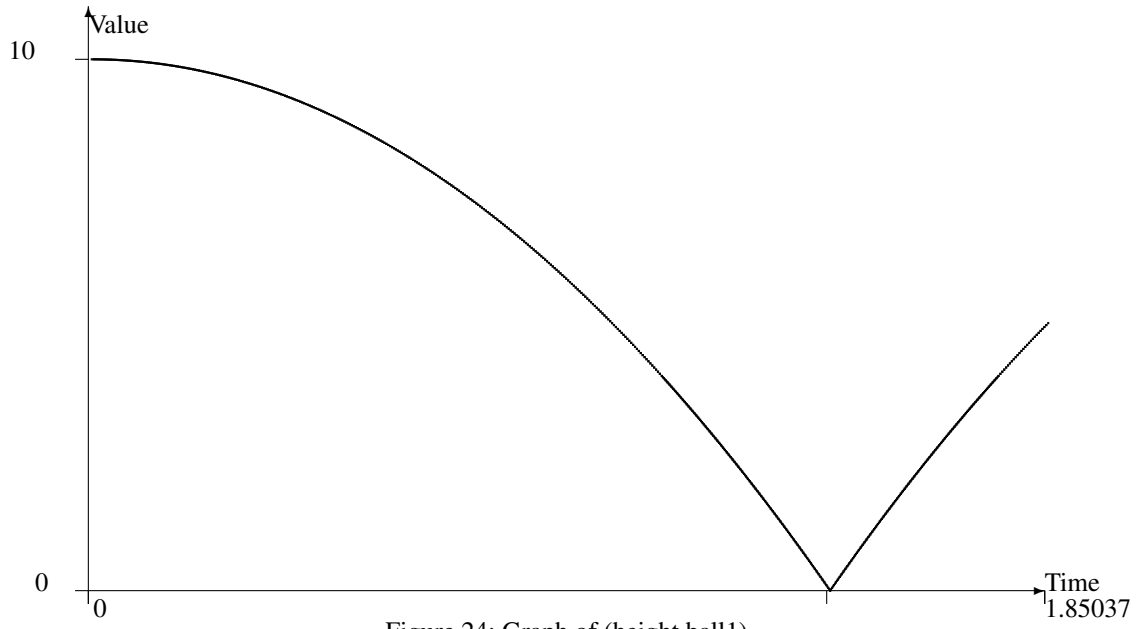


Figure 24: Graph of (height ball1).

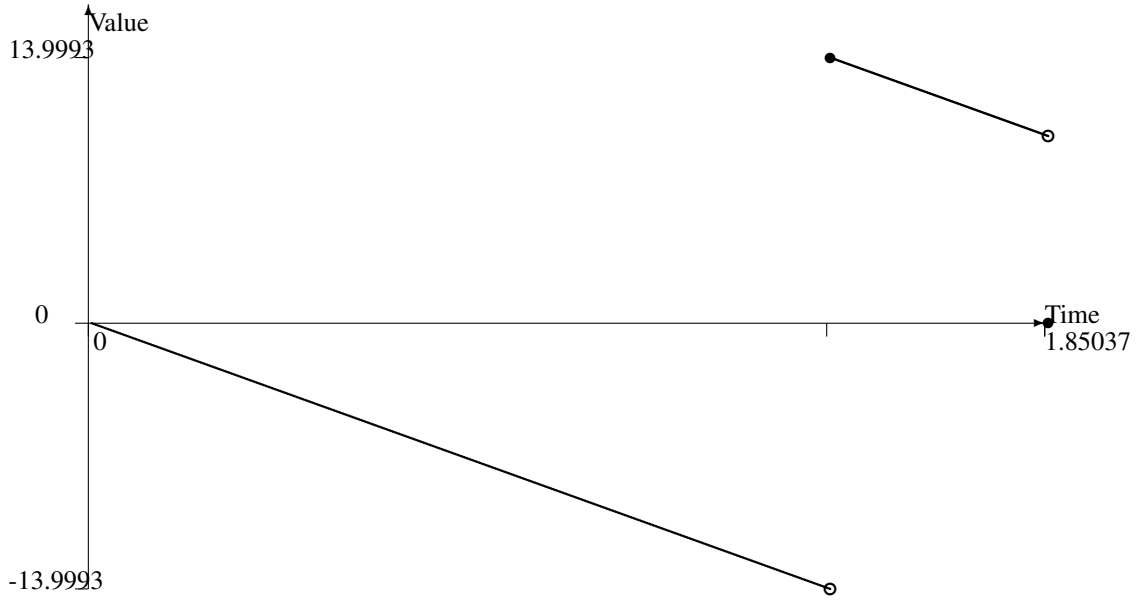


Figure 25: Graph of (velocity ball1).

parameters as described by Savas et al. are constrained with an upper bound and a lower bound in the domain file. The main example used by Savas et al. is a domain which includes the action of withdrawing cash from an ATM (referred to as the *cashpoint* example). Choosing the value of the withdrawal is a numeric control parameter in this action. Without this numeric control parameter the amount of withdrawal would have to be

modelled by a set of actions discretising the real-valued parameter, or by introducing actions to increase and decrease the amount of withdrawal by a fixed step size. In either case, the amount of withdrawal is restricted to a discretisation and the number of actions is increased.

In this section we briefly describe how Savas et al. extended PDDL2.1 to include control parameters, and later we show the extension to our SMT encoding for the domains with control parameters. Note that while previous work on control parameters has focused on PDDL2.1, our encoding treats control parameters in any PDDL2.1 or PDDL+ domain. We use the *cashpoint* domain as a running example throughout the section.

6.1 Cashpoint Problem

In the cashpoint problem, suppose that we want to go to a pub. Figure 26 shows the initial state and goal of the cashpoint problem. Initially we are at home and have £2 in our pocket. The aim of the problem is to be at the pub with £20 in our pocket and have purchased snacks on the way to the pub.

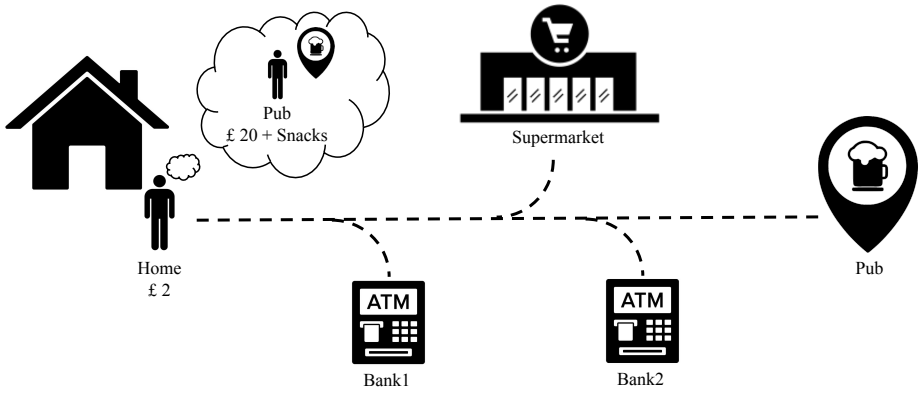


Figure 26: Initial state and goal of the cashpoint problem. The person is initially at home with £2 and the goal is to be at the pub with snacks and £20.

The amount of money that we withdraw from the cash machine is defined as a control parameter whose value is chosen by the planner. The proposed PDDL domain with control parameters is an extended version of PDDL2.1 in which durative actions can include an additional field for control parameters.

Figure 27 shows a fragment of the domain for this problem. The action $(withdraw\_money)$ is a durative action with the control parameter $(?cash - number)$. In this action, the control parameter is used in the effects of the action and not in the conditions. As the effect of this action, the numeric variable $(inpocket)$ is increased by the value of the control parameter. Similar to that the numeric variable $(maxwithdraw\ ?b)$, where $?b$ is the location that we withdraw money from, has been decreased by the same value.

Figure 28 shows an example problem file for the cashpoint domain. In the problem file for this example there are five locations. Two banks, each of which has a maximum amount of £200 to withdraw, a supermarket where snacks can be purchased, the pub, and the home. Later, we use this example to show the encoding of control parameters.

6.2 Encoding of Domains with Control Parameters

In this section we describe in detail how we extend our previous encoding described in Section 4 to handle actions with control parameters. First, we discuss the modifications over the definition of a single happening by introducing control parameters and later we extend these changes to the constraints between happenings.

Following Savas et al. (2016), control parameters are only defined for durative actions. An action with a control parameter, $a \in A$, is defined as:

$$a := \langle pre_a, eff_a, dur_a, cparam_a \rangle$$

```
(:durative-action withdraw_money
:parameters (?p - person ?a - location)
:control (?cash - number)
:duration (= ?duration 2)
:condition (and (over all (at ?p ?a))
                (at start (>= ?cash 5))
                (at start (<= ?cash 100))
                (at start (available))
                (at start (canwithdraw_money ?a))
                (at start(>= (maxwithdraw ?a) ?cash))
            )
:effect (and
                (at start (decrease (maxwithdraw ?a) ?cash))
                (at start (increase (inpocket ?p) ?cash))
                (at start (not (available)))
                (at end (available))
        ))


(:durative-action buy_snacks
:parameters (?a - location ?p - person)
:duration (= ?duration 1)
:condition (and     (at start (at ?p ?a))
                    (over all (at ?p ?a))
                    (at start (available))
                    (at start (>= (inpocket ?p) 5))
                    (at end (>= (inpocket ?p) 0))
                    (at start (canbuy ?a)) )
:effect (and    (at start (decrease (inpocket ?p) 5))
                (at start (not (available)))
                (at end (gotsnacks))
                (at end (available))
        ))

(:durative-action buy_drinks
:parameters (?a - location ?p - person)
:duration (= ?duration 1)
:condition (and     (at start (at ?p ?a))
                    (over all (at ?p ?a))
                    (at start (available))
                    (at start (>= (inpocket ?p) 4))
                    (at end (>= (inpocket ?p) 0))
                    (at start (canbuy ?a)) )
:effect (and    (at start (decrease (inpocket ?p) 4))
                (at start (not (available)))
                (at end (gotdrinks))
                (at end (available))
        ))
```

Figure 27: Three operators from the cashpoint domain, for withdrawing money, buying snacks, and buying drinks.

where $cparam_a$ represents a finite set of numeric control parameters, where each $d^a \in cparam_a$ has a domain $dom(d^a)$. In order to use the control parameters in our encoding, we add a new set of variables to the encoding of a happening explained in Section 4.1:

$$D_t := \{d_t^a \in cparam_a, \forall a \in A\}$$

where each $d_t^a \in \mathbb{R}$ describes the value of a control parameter. The encoding of a happening at time t is therefore defined as the tuple:

$$x_t := \left\langle \begin{array}{l} time, P_{0,t}, \ldots, P_{B,t}, P_{B+1,t}\, V_{0,t}, \ldots, V_{B,t}, V_{B+1,t} \\ D_t, E_{0,t}, \ldots, E_{B,t}, Ps_t, A_t, flow_{V_t}, dur_{Ps_t} \end{array} \right\rangle$$

By introducing the control parameter variables, the constraints over a happening are extended in the following way. As explained before, the control parameters are bounded in the domain file. First, two sets are defined U_d and L_d. The members of set U_d are constants which define the upper bounds of control parameters

```
   (:objects
        emre - person
        pub supermarket home - location
        bank1 bank2 - location)

   (:init
        (at emre home)
        (canbuy supermarket)
        (canwithdraw_money bank1)
  (available)
        (= (maxwithdraw bank1) 200)
        (= (maxwithdraw bank2) 200)
        (= (inpocket emre) 2)
 )

 (:goal (and
        (gotsnacks)
        (at emre pub)
        (>= (inpocket emre) 120)
    ))
)
```

Figure 28: An example problem file for the cashpoint domain.

at each happening (since these upper bounds will not change in different happenings, we do not need to define different upper bounds for each happening). L_d represents the set of lower bounds for the control parameters. Figure 29 shows the set of constraints within a happening in the case of having control parameters. The changes are shown in bold compared to Figure 8.

Within a happening, the structure of most of the constraints remain the same. However we need to add the constraints associated with the lower and upper bounds of control parameters (shown in equations $H15$ and $H16$). These constraints state that when the action is applied, the control parameter is bounded by its constant upper and lower bounds $u_d \in U_d$ and $l_d \in L_d$, respectively. In addition, control parameters can appear in the conditions or effect of a durative action. Therefore $H9$ and $H10$ are also affected by control parameters. The control parameter variables $d_t \in D_t$ can be used directly in these constraints.

Adding control parameters to durative actions also affects the encoding of the constraints between the happenings explained in section 4.2. Figure 30 shows these equations. The changes compared to Figure 10 are shown in bold. The only constraint added to the set of constraints between the happenings is *control parameter support* constraint ($P12$). This constraint ensures that the value of a control parameter stays constant during the execution of the action. In P12, dur_{a_i} refers to the duration variable of the action a containing the control parameter.

6.3 Encoding of the cashpoint problem

Considering the domain shown above, we introduce a variable d_t^a for each happening and control parameter associated with a durative action. In this domain we have two durative actions with a control parameter:

$$(withdraw\_money\ emre\ bank1), (withdraw\_money\ emre\ bank2)$$

Therefore two new variables are added in each happening:

$$cash_t^{(withdraw\_money\ emre\ bank1)}$$
$$cash_t^{(withdraw\_money\ emre\ bank1)}$$

Proposition and real variable support

H1. $\bigwedge_{p\in P} p_{1,t} \rightarrow (p_{0,t} \vee \bigvee_{e\in E|p\in eff_e^+} e_{0,t} \vee \bigvee_{a_t\in A|p\in eff_a^+} a_t)$

H2. $\bigwedge_{p\in P} \neg p_{1,t} \rightarrow (\neg p_{0,t} \vee \bigvee_{e\in E|p\in eff_e^-} e_{0,t} \vee \bigvee_{a\in A|p\in eff_a^-} a_t)$

H3. $\bigwedge_{i=1}^{B} \bigwedge_{p\in P} p_{i+1,t} \rightarrow (p_{i,t} \vee \bigvee_{e\in E|p\in eff_e^+} e_{i,t})$

H4. $\bigwedge_{i=1}^{B} \bigwedge_{p\in P} \neg p_{i+1,t} \rightarrow (\neg p_{i,t} \vee \bigvee_{e\in E|p\in eff_e^-} e_{i,t})$

H5. $\bigwedge_{v\in V}(\bigwedge_{a\in A|v\in eff_a^{num}} \neg a_t \wedge \bigwedge_{e\in E|v\in eff_e^{num}} \neg e_{0,t}) \rightarrow (v_{i+1,t} = v_{i,t})$

H6. $\bigwedge_{i=1}^{B} \bigwedge_{v\in V}(\bigwedge_{e\in E|v\in eff_e^{num}} \neg e_{i,t}) \rightarrow (v_{i+1,t} = v_{i,t})$

Event preconditions and effects

H7. $\bigwedge_{i=0}^{B} \bigwedge_{e\in E} e_{i,t} \leftrightarrow pre_e(P_i \cup V_i)$

H8. $\bigwedge_{i=0}^{B} \bigwedge_{e\in E} e_{i,t} \rightarrow eff_e(P_{i+1} \cup V_{i+1})$

Action preconditions and effects

H9. $\boldsymbol{\bigwedge_{a\in A} a_t \rightarrow pre_a(P_{0,t} \cup V_{0,t})}$

H10. $\boldsymbol{\bigwedge_{a\in A} a_t \rightarrow eff_a(P_{1,t} \cup V_{1,t})}$

Process triggering

H11. $\bigwedge_{ps\in Ps} ps_t \leftrightarrow pre_{ps}(P_{B+1,t} \cup V_{B+1,t})$

H12. $\bigwedge_{ps\in Ps} dur_{ps_t} >= 0$

H13. $\bigwedge_{ps\in Ps} ps_t \leftrightarrow (dur_{ps_t} > 0)$

Action mutexes

H14. $\bigwedge_{a\in A} \bigwedge_{a'\in A|a\nparallel a'} (\neg a_t \vee \neg a'_t)$

Bounds on control parameters

H15. $\boldsymbol{\bigwedge_{d_t^a\in D_t} a_t \rightarrow d_t^a <= u_d}$

H16. $\boldsymbol{\bigwedge_{d_t^a\in D_t} a_t \rightarrow d_t^a >= l_d}$

Figure 29: Encoding of a PDDL+ happening with control parameters to SMT.

For the first action, $(withdraw\_money\ emre\ bank1)$, there are two numeric variables affected by the associated control parameter. These are are $(maxwithdraw\ bank1)$ and $(inpocket\ emre)$. Constraint $H10$ is modelled as the following constraint, which illustrates the effect of the control parameters on the numeric variables.

$$\begin{gathered}(withdraw\_money\ emre\ bank1)_{sta,t} \rightarrow \\ (maxwithdraw\ bank1)_{1,t} = (maxwithdraw\ bank1)_{0,t} - cash_t^{(withdraw\_money\ emre\ bank1)}) \\ \wedge \\ (withdraw\_money\ emre\ bank1)_{sta,t} \rightarrow \\ (inpocket\ emre)_{1,t} = ((inpocket\ emre)_{0,t} + cash_t^{(withdraw\_money\ emre\ bank1)})\end{gathered}$$

Instance description

P1. $I(P_{0,1} \cup V_{0,1})$

P2. $G(P_{B+1,n} \cup V_{B+1,n})$

P3. $time_0 = 0$

P4. $\bigwedge_{i=2}^{n} time_i \geq time_{i-1} + \varepsilon$

Proposition support

P5. $\bigwedge_{i=2}^{n} \bigwedge_{p \in P} p_{0,i} \rightarrow p_{B+1,i-1}$

P6. $\bigwedge_{i=2}^{n} \bigwedge_{p \in P} \neg p_{0,i} \rightarrow \neg p_{B+1,i-1}$

Invariants

P7. $\bigwedge_{i=2}^{n} \bigwedge_{ps \in Ps} ps_{i-1} \rightarrow dur_{ps_i} = dur_{ps_{i-1}} - time_i + time_{i-1}$

P8. $\bigwedge_{i=1}^{n-1} \bigwedge_{ps \in Ps} ps_i \leftrightarrow pre_{\leftrightarrow ps_i}$

P9. $\bigwedge_{i=1}^{n-1} \bigwedge_{e \in E} \neg pre_{\leftrightarrow e_i}$

Continuous change on real variables

P10. $\bigwedge_{i=1}^{n-1} \bigwedge_{v \in V} flow_{v_i} = \int_{time_i}^{time_{i+1}} \sum_{ps \in Ps} eff_{\leftrightarrow ps}^{num}(V_i) dt$

P11. $\bigwedge_{i=2}^{n} \bigwedge_{v \in V} (v_{0,i} = v_{B+1,i-1} + flow_{v,i-1})$

Control parameter support

P12. $\boldsymbol{\bigwedge_{i=1}^{n-1} \bigwedge_{d_t^a \in D_t} (dur_{a_i} > 0) \rightarrow (d_i^a = d_{i+1}^a)}$

Figure 30: Encoding of PDDL+ planning problem with control parameters $\Pi+$ to SMT.

According to the conditions of $withdraw\_money$ in the domain, the upper bound u_d on the control parameter is 100 and the lower bound l_d is 5. Using these bounds we can write the equations $H15$ and $H16$ as :

$$(withdraw\_money\ emre\ bank1)_{sta,t} \rightarrow cash_t^{(withdraw\_money\ emre\ bank1)} >= 5$$
$$(withdraw\_money\ emre\ bank1)_{sta,t} \rightarrow cash_t^{(withdraw\_money\ emre\ bank1)} <= 100$$

The plan found by the SMTPlan for this domain and problem file is shown in Figure 31. The solution found by the SMT solver is not guaranteed to be optimal with respect to time. This is demonstrated by the plan shown in Figure 31, in which there is a 2-second gap between the end of the first withdraw action and the beginning of the second. This is because the only constraint is that the two happenings are separated by at least ε, and the SMT solver is left to choose any valid value. It is possible to include a metric for the SMT solver to optimise, for example, the time point of the final happening. However, this does not guarantee optimality with respect to plan duration, as increasing the bound on the number of happenings might lead to a solution with shorter duration.

```
0.0: (goto emre home bank1) [5.0]
7.0: (withdraw_money emre bank1) [2.0] (?cash [24.0])
11.0: (withdraw_money emre bank1) [2.0] (?cash [99.5])
15.0: (goto emre bank1 supermarket) [5.0]
22.0: (buy_snacks supermarket emre) [1.0]
25.0: (goto emre supermarket pub) [5.0]
```

Figure 31: The plan found by the SMTPlan for the example cashpoint problem.

Table 4 shows part of a satisfying assignment to an encoding of a problem in the cashpoint domain. The assignment corresponds to the plan shown in Figure 31, and displays only the second and third happenings. The second happening occurs 7 time units after starting the plan, and the duration of interval between the two happenings is 2 time units.

x_2	x_3
$t_2 := 7$	$t_3 := 9$
$(available)_{0,2} := 1$ $(available)_{1,2} := 0$	$(available)_{0,3} := 0$ $(available)_{1,3} := 1$
$(maxwithdraw\ bank1)_{0,2} := 200$ $(maxwithdraw\ bank1)_{1,2} := 176$ $(inpocket\ emre)_{0,2} := 0$ $(inpocket\ emre)_{1,2} := 24$	$(maxwithdraw\ bank1)_{0,3} := 176$ $(maxwithdraw\ bank1)_{1,3} := 176$ $(inpocket\ emre)_{0,3} := 24$ $(inpocket\ emre)_{1,3} := 24$
$(withdraw\_money\ emre\ bank1)_{sta,2} := 1$ $(withdraw\_money\ emre\ bank1)_{end,2} := 0$	$(withdraw\_money\ emre\ bank1)_{sta,3} := 0$ $(withdraw\_money\ emre\ bank1)_{end,3} := 1$
$(withdraw\_money\ emre\ bank1)_{dur,2} := 2$	$(withdraw\_money\ emre\ bank1)_{dur,3} := 0$

Table 4: Assignment to numeric and propositional variables associated with the $withdraw\_money$ action in the cashpoint domain. Boolean variables that are true are assigned as 1 and the ones that are false are 0.

7. Experimental Evaluation

In this section we present our experimental results in four parts.

1. In Section 7.1 we compare the performance of SMTPlan against other PDDL+ planners on PDDL+ benchmarks for hybrid systems domains. We aim to show that in problems with non-linear continuous change, and a small number of happenings, SMTPlan performs very well.

2. In Section 7.2 we investigate the scalability of SMTPlan in PDDL+ domains in more detail.

3. In Section 7.3 we compare SMTPlan with temporal planners from the International Planning Competition (IPC) on a set of temporal planning benchmarks used in the IPC. In the experiment we explore the limitations inherent in the Satisfiability approach in terms of scalability with respect to the size of the discrete state-space and number of happenings.

4. Finally, in Section 7.4 we evaluate the effect of control parameters on our encoding on several PDDL2.1 and PDDL+ domains.

7.1 PDDL+ Benchmarks

We use our encoding to solve PDDL+ planning problems with a iterative deepening technique, widely used in SAT-based planning approaches (Nabeshima, Iwanuma, & Inoue, 2002; Rintanen et al., 2006). The top-level algorithm iteratively encodes and solves SMT encodings, solving the planning problem for horizon lengths 2,3,4,.... In this case the horizon length corresponds to number of happenings. If a formula is found satisfiable, then a plan has been found and the planner terminates. If a formula is found unsatisfiable, then an encoding is made for the next shortest horizon length. The SMT solver we use is z3 (De Moura & Bjørner, 2008). The algorithm is described more fully in Section 4.4.

For all experiments we use an initial happening bound of 2 (assuming the goal to not hold in the intial state) a step size of 1, and a bound on the length of the event cascade of 2. The bound on the event cascade is known to be the longest cycle-free chain of events possible across all of the benchmark domains.

We compare our approach (called SMTPlan) against existing PDDL+ planners UPMurphi (Della Penna et al., 2012), DiNo (Piotrowski, Fox, Long, Magazzeni, & Mercorio, 2016), and with dReach (Bryce, Gao, Musliner, & Goldman, 2015) using the SMT solver dReal (Gao, Avigad, & Clarke, 2012), on domains both with and without events[4]. We use the *generator* and *car* domains (Bogomolov et al., 2014). The experiments were run using 8GB of RAM and a 30 minute timeout. UPMurphi and DiNo used a time discretisation of 0.1 seconds for all experiments. All test domains and problems are available at: *kcl-planning.github.io/SMTPlan*.

4. We considered domains without events to show a comparison with dReach, which does not handle domains including events.

The generator domain is a PDDL+ benchmark problem that revolves around refueling a diesel-powered generator, which has to run for a given duration without overflowing or running dry. To test scalability the number of tanks is increased while decreasing the initial fuel level, so that all tanks are required.

We consider four versions of this domain: linear, simplified-nonlinear (the same used in (Bryce et al., 2015)), nonlinear with events, and the Torricelli nonlinear (Howey & Long, 2003). Note that the latter version uses the Torricelli's Law, and hence the fuel level in a refueling tank (V_{fuel}) is calculated by:

$$V_{fuel} = (-kt_r + \sqrt{V_{init}})^2 \qquad t_r \in \left[0, \frac{\sqrt{V_{init}}}{k}\right] \tag{6}$$

where V_{init} is the initial volume of fuel in the tank, k is the fuel flow constant (which depends on gravity, size of the drain hole, and the cross-section of the tank), and t_r is the time of refueling (bounded by the fuel level and the flow constant). An example of plan found by SMTPlan for the Torricelli nonlinear generator (Fuel level 960, Generator capacity 990) is shown in Figure 32:

```
0.0: generate     [1000.0]
959.0: refuel_tank1 [10.0]
959.0: refuel_tank2 [10.0]
```

Figure 32: Plan for a problem in the Toricelli Generator domain.

The car domain is another PDDL+ benchmark (Fox & Long, 2006) where a vehicle has to cover a given distance and have a zero velocity at the end, and the actions available are accelerate and decelerate that increments or decrements by 1 the current velocity, respectively. To test scalability, the bound on maximum acceleration/deceleration is increased.

Our results for solvable instances are reported in Table 5. On linear and non-linear domains, SMTPlan outperforms all other planners in time to solve and in number of instances solved. In the Generator domain (linear and with nonlinear with events), SMTPlan finds solutions more quickly than DiNo, but scales more poorly in the larger instances. However, in all domains, SMTPlan finds solutions relatively quickly. For these domains, the number of happenings required is small, thus the minimal SMT encoding required to solve the problem is also small. The iterative deepening algorithm is able to reach a satisfiable encoding, and produce a plan very quickly. The offline computation with PIRANHA is required only once per domain, and in all cases requires 0.3 seconds or less. Performance with all planners is worse on the domain with events, due to the presence of the generator overflow event, which must be avoided in a valid plan.

dReach also performs iterative deepening, but performs more poorly. This is due to the semantics of dReach; in the dReach domain and problem description, each mode of continuous change must be explicitly defined, and the number of modes increases exponentially with the number of processes and durative actions (eg. the files for 1, 2, 3 and 4 tanks problems are respectively 91, 328, 1350, 5762 lines long). Furthermore, the bound is not on the number of happenings, but on the number of mode changes, which does not allow for parallel execution of actions. In contrast to SMTPlan, dReach does not perform integration and differentiation during encoding. Instead it relies upon the more expressive logic of the internal SMT solver, dReal. As a result, it is unable to use other SMT solvers.

We also compare our encoding directly against dReach as reported in prior work (Bryce et al., 2015): reporting times to solve only the encoding of a minimal step plan for each instance (Table 6). These are not the times required to solve a PDDL+ instance, but a direct comparison of encodings on satisfiable problems. We find the encodings exhibit similar performance in the car domain. However, we find the SMTPlan encoding scales far better on the generator problem, for the reasons discussed above.

Our results for unsolvable instances are shown in Table 7. SMTPlan and dReach can only prove unsolvability up to an upper bound on the number of happenings. Here we prove plan non-existence for domains which have a tight deadline, and where each ground action can only be applied a finite number of times. We observe that both totally ordered planning approaches perform well proving unsolvability in the car domain.

There are few choices of symbolic plan in this domain, leaving only the timing of the happenings and numeric constraints to be solved. Both SMTPlan and dReach solve these constraints very quickly. However, for PDDL+ problems in general, without deadlines and with repeatable actions, proving unsolvability is difficult through totally ordered planning with iterative deepening.

Domain	Tool	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Generator linear	SMTPlan	0.03	0.01	0.01	0.02	0.02	0.04	0.09	0.18	0.37	0.86	2.07	5.90	17.10	63.80	-
	dReach	2.04	4.16	6.87	627.77	-	-	-	-	-	-	-	-	-	-	-
	UPMurphi	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
	DiNo	17.57	23.71	32.2	45.87	53.55	66.34	78.51	92.54	108.84	124.35	136.08	152.17	169.59	188.89	208.67
Generator nonlinear	SMTPlan	0.01	0.02	0.03	0.07	0.18	0.41	1.01	2.47	5.7	13.65	33.76	87.08	255.27	797.21	-
	dReach	1.51	4.62	-	-	-	-	-	-	-	-	-	-	-	-	-
	UPMurphi	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
	DiNo	22.29	34.55	.	.	.	.	.	.	.	.	.	.	.	.	.
Generator nonlin. events	SMTPlan	0.02	0.07	0.37	1.27	6.36	33.72	256.06	-	-	-	-	-	-	-	-
	dReach	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
	UPMurphi	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
	DiNo	22.64	30.59	55.75	93.58	146.46	216.93	310.43	422.71	541.24	716.27	-	-	-	-	-
Generator Torricelli	SMTPlan	0.01	0.02	0.05	0.13	0.25	0.56	1.42	3.35	8.39	19.83	47.34	111.79	225.31	622.79	1590
	dReach	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
	UPMurphi	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
	DiNo	23.7	79.26	.	.	.	.	.	.	.	.	.	.	.	.	.
Car	SMTPlan	0.02	0.02	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.02	0.02	0.03	0.03
	dReach	1.44	1.82	1.99	-	-	-	-	-	-	-	-	-	-	-	-
	UPMurphi	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
	DiNo	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-

Table 5: Results in seconds for solvable instances. Instance numbers correspond to number of tanks (generator) and number of acceleration steps (car). Abbrev.: '-': tool still running after 30 minutes, '.': tool ran out or memory, 'x': tool cannot handle the problem.

Domain	Tool	1	2	3	4	5	6	7	8
Generator linear	SMTPlan	0.00	0.01	0.01	0.01	0.01	0.02	0.02	0.02
	dReach	2.73	13.47	104.61	695.70	-	-	-	-
Generator nonlinear	SMTPlan	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01
	dReach	10.42	1685.35	-	-	-	-	-	-
Car	SMTPlan	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	dReach	0.77	0.76	0.76	0.76	0.76	0.76	0.77	0.76

Table 6: Results in seconds for minimal step encoding required to solve each instance. Abbrev.: '-': tool still running after 30 minutes.

7.2 Free Fall and Generator

Two experiments were designed to explore the performance of SMTPlan on PDDL+ domains in more detail.

The first experiment evaluates the scaling of SMTPlan with respect to the number of objects in the problem instance. The *Free Fall* problem was extended to consider problem instances with more than one ball. We considered two scenarios for these problem instances, catching one ball or all of the balls. In both scenarios we create cases that include 1, 25, 50, 100, and 200 balls. All of the balls have the same initial height and can be caught at the same height. The results for this scenario are shown in Table 8.

Both UPMurphi and DiNo discretise time to perform forward search. Considering the event *bounce* has a precondition of $(<= (height ?b) 0.001)$, the smallest time discretisation that can catch this event has the time step of 0.0001. However as illustrated by Table 8, both UPMurphi and DiNo quickly run out of memory with this discretisation.

Domain	Tool	1	2	3	4	5	6	7	8
Generator linear	SMTPlan	0.01	0.02	0.16	2.84	390.86	-	-	-
	dReach	2.57	189.94	-	-	-	-	-	-
	UPMurphi	0.90	29.42	-	-	-	-	-	-
Generator nonlinear	SMTPlan	0.01	1.95	33.48	-	-	-	-	-
	dReach	2.43	212.43	-	-	-	-	-	-
	UPMurphi	-	-	-	-	-	-	-	-
Generator nonlin. events	SMTPlan	0.02	18.58	21.83	-	-	-	-	-
	dReach	x	x	x	x	x	x	x	x
	UPMurphi	-	-	-	-	-	-	-	-
Generator Toricelli	SMTPlan	0.03	2.06	19.57	-	-	-	-	-
	dReach	x	x	x	x	x	x	x	x
	UPMurphi	-	-	-	-	-	-	-	-
Car	SMTPlan	0.68	0.02	0.00	0.00	0.00	0.00	0.00	0.01
	dReach	0.67	0.50	0.62	0.45	0.58	0.57	0.49	0.65
	UPMurphi	36.01	445.23	-	-	-	-	-	-

Table 7: Results in seconds for unsolvable instances. Instance numbers correspond to number of tanks (generator) and number of acceleration steps (car). Abbrev.: '-': tool still running after 30 minutes, 'x': tool cannot handle the problem.

Free Fall	Tool	1	25	50	150	200
Catch one ball	SMTPlan	0.06	0.1	0.2	0.46	1.21
	UPMurphi	.	.	.	.	.
	DiNo	.	.	.	.	.
Catch all balls	SMTPlan	NA	2.31	8.54	32.81	118.15
	UPMurphi	NA	.	.	.	.
	DiNo	NA	.	.	.	.

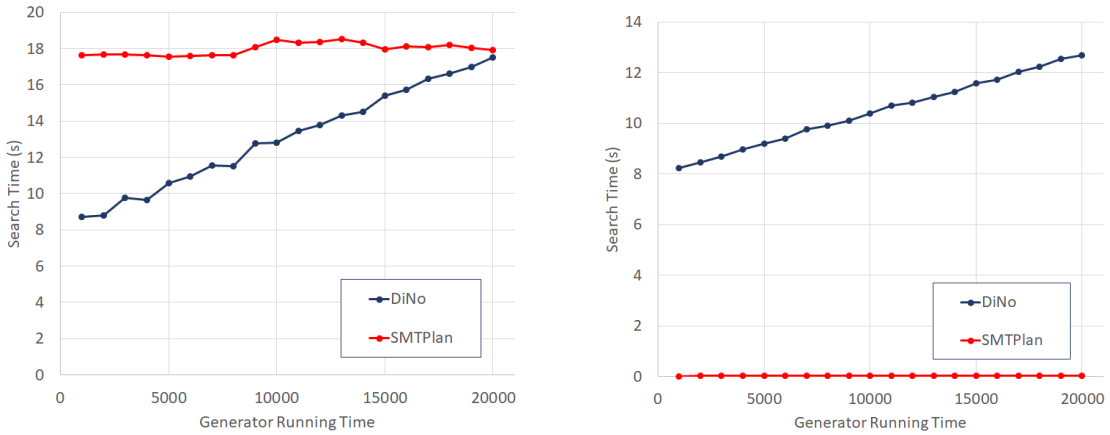
Table 8: Time to solve in seconds for instances of the Free Fall domain. Instance numbers correspond to number of balls (1, 25, 50, 100 and 200). Abbrev.: 'NA': not applicable, '.': tool ran out of memory.

As we increase the number of balls in the problem file, the search space grows exponentially. However, as shown in Table 8, SMTPlan is able to scale to all of the problems. This derives from the fact that all the balls are at the same height and need to be caught at the same height. The number of happenings that is needed to solve all the problems is the same. For this reason, in contrast to the forward search planners, SMTPlan scales well with respect to the number of objects on these problems. While it is true that increasing the number of objects often will also increase the minimum number of required happenings, the iterative deepening search of SMTPlan will attempt to perform parallelism of actions where possible.

To assess the scaling of SMTPlan with respect to the number of happenings, we generated new instances of the *Free Fall* problem which require increasing numbers of happenings. SMTPlan was used to generate plans, and the results are shown in Table 9. The performance of SMTPlan remains excellent for happenings 2 to 5, and then degrades very quickly after 5 happenings. The exact number of happenings that can be handled by SMTPlan depends strongly upon the domain, but the phase transition between solvable and unsolvable

		Number of Happenings				
Domain	Tool	2	3	4	5	6
Free Fall	SMTPlan	0.2	0.03	0.14	5.31	-

Table 9: Time to solve in seconds for instances of the Free Fall domain. The number of happenings needed to solve the instances increase gradually. Abbrev.: '-': tool still running after 30 minutes.



(a) (b)

Figure 33: Generator domain with (a) nonlinear change and events and (b) only linear change, with increasing time horizon.

problems is typically very small. Experiments on temporal problems are shown in Section 7.3, illustrating the effect of this scaling behaviour in PDDL2.1 domains.

The second experiment on PDDL+ domains tested the scaling performance of SMTPlan with respect to longer time horizons. A number of problems were generated for the *generator* domain in which the capacity and the running time of the generator increases gradually. In the benchmark problems the running time of the generator is 1,000 time units. We created problems where the generator's running time is increased up to 20,000 with increments of 1,000 time units. To test the effects of the time horizon on the planner, we also changed the amount of the fuel in the refuelling tank, so that the number of times that we need to apply the re-fuelling action remains the same. In other words, just the length of the re-fuelling process has been extended. The problems were solved with both SMTPlan and DiNo. As shown in Figures 33a and 33b, the time taken by SMTPlan to find a plan is not related to the time horizon. However, as we increase the time horizon, DiNo needs a longer time to solve the problem. As a forward search planner, the duration of the plan greatly impacts the performance of DiNo, and impacts the choice of discretisation that it can use. On the other hand, since the number of the happenings are the same, increasing the plan duration does not affect SMTPlan.

7.3 Temporal Domains

To investigate the scaling performance of SMTPlan with respect to the size of the discrete search space, we also tested the SMTPlan on the domains that were used in the temporal track of the International Planning Competition (IPC) at ICAPS 2018. We compared the SMTPlan with the other planners that participated in this track. These planners are CP4TP (Furelos-Blanco & Jonsson, n.d.), TFLAP (Oscar Sapena, n.d.), TemPorAl (Cenamor, Vallati, Chrpa, de la Rosa, & Fernndez, n.d.), PopCorn (Savas, Edelkamp, nad Derek Long, & Magazzeni, n.d.) and OPTIC (A. Coles & Coles, n.d.). In total 10 domains were chosen. For each domain

the first 10 problem instances were passed to the planner to solve. Each planner was given 30 minutes and 8GB of memory to solve as many problems as possible. The results are shown in Table 10. Each number in the table indicates the number of problem instances that each planner has managed to solve. In each row the largest number of problems for that domain solved by a single planner is shown in bold.

As can be seen in the table, the performance gap between SMTPlan and temporal planners, on temporal planning problems, is large. While SMTPlan is highly effective on PDDL+ problems with nonlinear change and a small discrete search space, the large discrete search space for temporal planning problems is often insurmountable. These experiments show that SMTPlan scales poorly with respect to the number of happenings in comparison to forward search planners. One exception is the *Cushing* domain, in which SMTPlan performed the best. This domain has a high degree of required concurrency, and can be difficult for other planning approaches, such as forward search. In addition, it allows for a great amount of parallelism of actions, therefore allowing SMTPlan to scale easily to all of the problem instances considered.

		Planners					
Domain	N	SMTPlan	CP4TP	TFLAP	TemPorAl	PopCorn	OPTIC
Airport	10	1	9	9	**10**	3	3
Cushing	10	**10**	**10**	3	0	1	**10**
Floortile	10	0	2	3	**10**	0	0
Map Analyser	10	0	**10**	8	**10**	0	0
Parking	10	0	**10**	**10**	**10**	4	8
Quantum	10	1	**10**	8	**10**	5	8
Road Traffic	10	0	7	0	**10**	0	0
Sokoban	10	0	**6**	4	**6**	1	1
Trucks-time-strips	10	0	**10**	**10**	**10**	9	**10**
Total	100	12	74	55	**76**	23	40

Table 10: Number of problems that each planner could solve in 30 minutes for the domains from ICAPS 2018 temporal tracks.

7.4 Domains with Control Parameters

In this section we have designed a two sets of experiments to show how introducing control parameters affects the performance of SMTPlan. In the first set of experiments we used the *cashpoint* domain, solving problems with SMTPlan on two different variants of the domain: one including control parameter variables and the other one without. Following the setup of Savas et al. (2016), in the domain without control parameters we add a number of *withdraw_money* actions with fixed withdrawal values, namely $\{1, 5, 10, 20\}$. The upper and lower bounds on each control parameter is defined by the operator in the domain. The lower bound is zero and upper bound is 100 for the amount of withdrawal in the *withdraw_money* action. For both domains we considered a six sets of problem instances with an increasing number of bank locations (from 1 to 6). In each problem every bank must be visited to achieve the goal. Problems within each set increase in difficulty due to additional locations and goals.

The results for these experiments are shown in Table 11 and Figure 34. Table 11 shows the number of problems that could be solved with and without control parameters. By using control parameters, repeated applications of the same action can be avoided, lowering the number of required happenings. This allows SMTPlan to solve many more problems.

Figure 34 compares the time taken to solve problems that were solved both with and without control parameters. As we can see on the graph, the time to solve the problems without control parameters is generally much longer. However, in smaller problems the overhead of introducing the control parameter variables and constraints leads to longer solution times.

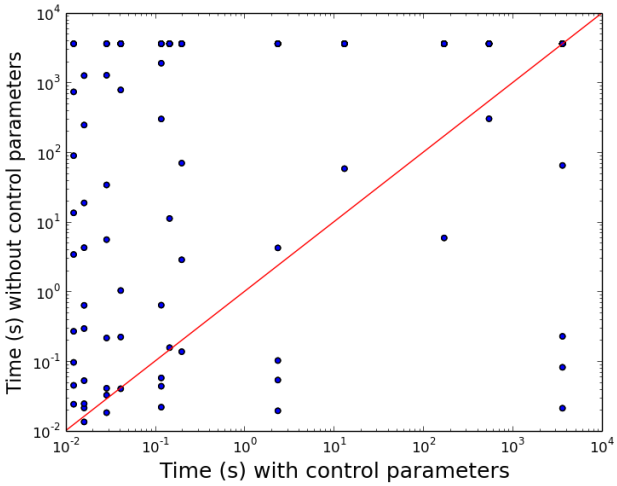


Figure 34: Comparison of time taken to solve problems in the *cashpoint* domain with and without control parameters.

No. Banks	Total Problems	Problems Solved	
		With CP	Without CP
1	60	40	16
2	60	30	13
3	60	20	8
4	60	10	6
5	60	10	4
6	60	0	0

Table 11: Comparison of number of problems in the *Cashpoint* domain that can be solved by SMTPlan with and without control parameters.

In the second set of experiments, we generated variants of PDDL+ and PDDL2.1 domains to include control parameters. For the PDDL+ domains, we created versions of the *Generator* domain in which the refuel rate is a control parameter, and a variant of the *Car* domain in which the acceleration (and deceleration) is a control parameter. For the PDDL2.1 domains we created variants of *Car* (without events and processes, called *Car-cp*) and *Satellite* in which the duration of actions is variable. In *Car-cp* the *moving* action has variable duration, but acceleration and deceleration must still be performed by multiple actions. In *Satellite* the *turn_to* action has variable duration.

Defining actions with variable duration is part of PDDL2.1 semantics, but can be thought of as a kind of control parameter. Note that in actions with continuous effects, including variable duration in the action can lead to non-linear change.

SMTPlan was used to solve problems and the results are shown in Table 12. For the domains in which control parameters are used to vary the effect of actions, SMTPlan performs well. This is due to the reduction in the number of happenings that results from avoiding repeated application of an action, such as *withdraw_money*. As shown by Table 9, the number of happenings can have an large impact on SMTPlan's performance. In the domains in which action duration is variable, the effects are less clear. As SMTPlan scales constantly with respect to the horizon of the plan, it is only in cases where a variable action duration

Domain	Tool	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Generator linear	SMTPlan	0.02	0.02	0.03	0.04	0.08	0.19	0.58	2.70	19.08	333.25	-	-	-	-	-
Generator nonlinear	SMTPlan	0.02	0.03	0.05	0.11	0.26	0.66	1.58	3.84	9.35	23.04	53.52	128.5	289.4	669.1	1548
Generator nonlin. events	SMTPlan	0.04	0.03	0.06	0.16	0.76	7.12	168.37	-	-	-	-	-	-	-	-
Generator Torricelli	SMTPlan	0.06	0.03	0.06	0.14	0.34	0.87	2.40	7.27	31.73	392.58	-	-	-	-	-
Car	SMTPlan	0.03	0.02	0.02	0.02	0.02	0.02	0.02	0.02	0.02	0.02	0.02	0.03	0.02	0.03	0.02
Satellite (variable durations)	SMTPlan	0.08	0.021	28.30	55.32	-	-	-	-	-	-	-	-	-	-	-
Car-cp (variable durations)	SMTPlan	0.02	0.03	0.05	0.11	0.26	0.66	1.58	3.84	9.35	23.04	53.52	128.5	289.4	669.1	1548

Table 12: Time to solve instances with control parameters. Results in seconds for solvable instances. Abbrev.: '-': tool still running after 30 minutes.

decreases the required number of happenings that SMTPlan's performance will be affected. Thus, the effect of adding actions with variable duration is dependent upon the domain.

8. Related Work

In this section we discuss the related work. First in section 8.1 we describe the preceding work in planning as satisfiability, and then in section 8.2 we place our work with resepect to other approaches to planning in hybrid domains.

8.1 Planning as Satisfiability

Planning as Satisfiability was pioneered by Kautz and Selman, beginning with a translation from Planning into propositional satisfiablity (SAT) (H. Kautz & Selman, 1992), and the planner SATPlan (H. A. Kautz et al., 2006). Since then there have been many contributions improving the effectiveness of planning as SAT (Rintanen, 1999; Huang, Chen, & Zhang, 2012), including alternate encodings of the state or transition relation; the embedding of additional planning-specific knowledge such as heuristic evaluation; or planning-specific improvements to the SAT solver. Many of these ideas are orthogonal to the choice of representation between SAT and SMT. There are relatively few approaches to planning as SMT. The dReach planner plans a subset of PDDL+ using the dReal SMT solver for ODEs, and TM-LPSAT uses an encoding of propositional and numeric variables with linear constraints solved by the LPSAT engine. However, neither of these approaches produce SMT problems that can be solved by general SMT solvers.

Lifted causal encodings, first introduced by Kautz et al. (1996), inspired by the lifted version of the SNLP causal link planner of McAllester and Rosenblitt (Mcallester & Rosenblitt, 1991) differ from the state-based encodings in that there is no proper notion of a state. The lifted encoding encodes a propositional planning problem as lifted SAT, which is then reduced to SAT. The encoding includes an assignment of action to plan steps, and the a valid causal ordering between plan steps. While it proved less effective in propositional planning with SAT, this work is applicable using the first-order expressions of SMT in a partially-ordered happening-based encoding.

Rintanen et al. have greatly advanced the state-in-the-art for planning as satisfiability, including new semantics for plan steps (Rintanen et al., 2006; Rintanen, 2017); embedding planning heuristics in the SAT solver (Rintanen, 2010a), such as the "helpful actions" heuristic in the planner Madagascar (Rintanen, 2010b), and other top-level search strategies that improve over the iterative deepening used by SATPlan and SMT-Plan (Rintanen, 2004). A shared purpose of these advancements is to help the SAT solver scale to the large discrete search space present in most planning problems. These approaches are orthogonal to the choice of encoding formalism (SAT or SMT), and can be applied directly in SMTPlan.

The planner ITSAT (Rankooh & Ghassem-Sani, 2015) is a SAT-based planner for non-numeric temporal planning problems. The planner extends the step semantics introduced by Rintanen et al. (2006) to temporally abstract the problem without losing the ability to express concurrent activities.

8.2 Planning in Hybrid Domains

Various techniques and tools have been proposed to deal with hybrid domains. ZENO (Penberthy & Weld, 1994) is a planner which can handle actions occurring over extended intervals of time. ZENO is able to reason about goals with deadlines, piece-wise linear continuous change, external events and to a limited extent, simultaneous actions. Similarly, the ScottyActivity planner (Fernandez-Gonzalez, Williams, & Karpas, 2018) is a forward search hybrid planner employing methods in convex optimization combined with relaxed plan graph heuristics.

More recent approaches in this direction have been proposed by Bologomov et al. (2014), where the close relationship between hybrid planning domains and hybrid automata is explored, and (Bryce et al., 2015) where hybrid domains are handled using SMT. dReach (Bryce et al., 2015), a planner for hybrid systems uses the dReal solver (Gao et al., 2012), a non-linear SMT solver that uses its own theory of ODEs. Input is provided in the language of dReach as opposed to PDDL+, and hybrid problems have to be manually encoded. This language can only handle a restricted subset of the language features contained in PDDL+. In particular, it cannot handle exogenous events.

More similar to the approach of SMTPlan is the planner TM-LPSAT (Shin & Davis, 2005). The planner is able to solve problems with atomic and durative actions, processes, events, and linear change. TM-LPSAT uses a happening-based encoding, containing propositional and numeric variables and linear constraints. This is solved by the LPSAT constraint engine. While SMTPlan is able to handle polynomial non-linear change, TM-LPSAT is restricted to continuous linear change; the LPSAT solver requires only linear constraints, and the encoding does not account for the zero-crossing problem introduced by non-linear change.

Many works have been proposed in the model checking and control communities to handle hybrid systems (Cimatti, Griggio, Mover, & Tonetta, 2015; Cavada et al., 2014; Cimatti, Mover, & Tonetta, 2012; Tabuada, Pappas, & Lima, 2002; Maly, Lahijanian, Kavraki, Kress-Gazit, & Vardi, 2013), including sampling-based planners (Karaman, Walter, Perez, Frazzoli, & Teller, 2011; Lahijanian, Kavraki, & Vardi, 2014), and formalisms for applying state-constraints on linear and non-linear numeric functions (Haslum et al., 2018). Another related direction is *falsification* of hybrid systems (Plaku, Kavraki, & Vardi, 2013) (i.e., guiding the search towards the error states, that can be easily cast as a planning problem). However, while all these works aim to address a similar problem, they cannot be used to handle PDDL+ models. Bogomolov et al. (2014, 2015) are working towards a formal translation between PDDL+ and standard hybrid automata, but so far only an over-approximation has been defined, which allows the use of those tools only for proving plan non-existence.

To date, the only viable approach to general PDDL+ planning is via discretisation. UPMurphi (Della Penna et al., 2012), which implements the discretise-and-validate approach, is able to deal with the full range of PDDL+ features. Discretise and validate works by first discretising time into small steps, solving the problem, and validating the result against the original continuous domain. If the plan is not valid with respect to the continuous semantics, then a finer discretisation is generated and the process iterates. However, UPMurphi performs blind search, which limits its scalability.

More recent work in this direction is the planner, DiNo. Similar to UPMurphi, DiNo is also based on the discretise-and-validate approach and uses the novel Staged Relaxed Planning Graph+ (SRPG+) heuristic (Piotrowski et al., 2016) to help cope with the scalability issues faced by UPMurphi.

Considering the related works mentioned above, the SMT encoding is able to capture all features of PDDL+ and works by directly translating standard PDDL+ domain and problem files. Furthermore, it correctly captures the *must* semantics of PDDL+ (which constrains how processes and events interact with each other and with actions). Also, SMTPlan models the precise semantics of ε-separation of effects and action preconditions (Fox & Long, 2006).

9. Conclusions and Future Work

In this paper we presented a new approach for planning for hybrid systems, based on encoding the planning problem as a Satisfiability Modulo Theories (SMT) formula. This is the first SMT encoding that can handle the whole set of PDDL+ features (including processes and events), and is implemented in the planner SMTPlan. SMTPlan not only covers the full semantics of PDDL+, but can also deal with non-linear polynomial continuous changes without discretization. This allows it to generate plans with non-linear dynamics that are correct-by-construction. The encoding is based on the notion of *happenings*, such that the variables of the SMT formula represent the state variables, and active events, actions, and processes within happenings. Between happenings there is only continuous numeric change. Constraints between variables in different happenings ensure any satisfying assignment to the SMT formula represents a valid plan trace.

The planning approach is an iterative deepening search, based on approaches to planning as SAT (Nabeshima et al., 2002; Rintanen et al., 2006). SMTPlan iteratively increases the bound on the number of happenings until a solution is found. The encoding is general and can be used with any SMT solver in the theory of quantifier-free nonlinear arithmetic.

Experimental results show that the approach dramatically outperforms existing work in finding plans for solvable PDDL+ problems, and is also efficient in proving plan-non-existence. The approach differs from traditional state-based search approaches to planning in that the performance is not affected by the duration of the plan, and only linearly by the number of objects in the problem instance. However, experimental results demonstrate that the scalability of the approach is limited by the size of the discrete search space.

We also showed how recent work on infinite-domain parameters, called *control parameters*, can be introduced into the SMTPlan encoding. The SMT-based approach to planning is a natural fit for planning with control parameters, and our experiments demonstrate the beneficial effects.

In our future work we intend to discover how approaches to planning as SAT, such as plan step semantics and embedded planning heuristics, can be used to help SMTPlan scale to problems with larger discrete state-spaces. In addition, we intend to explore the possibility of combined planning approaches, using state-based search to handle large discrete state-spaces, while using SMTPlan's novel approach for handling continuous numeric change. The restriction of SMTPlan to only polynomial nonlinear continuous change is a limitation of the CAS, which can only handle the indefinite integration of polynomials. In future work we plan to relax this restriction, and allow SMTPlan to handle a larger collection of nonlinear effects.

Acknowledgements

We thank Maria Fox, Derek Long, Alessandro Cimatti and Andrea Micheli for insightful discussions. This research was partly supported by Innovate UK grant 133549: *Intelligent Situational Awareness Platform*.

References

Aylett, R., Soutter, J. K., Petley, G. J., Chung, P. W., & Rushton, A. (1998). AI Planning in a Chemical Plant Domain. In *Proceedings of the 13th European Conference on Artificial Intelligence (ECAI)* (pp. 622–626).

Barrett, C., Stump, A., & Tinelli, C. (2010). The SMT-LIB Standard: Version 2.0. In *Proceedings of the 8th International Workshop on Satisfiability Modulo Theories.*

Barrett, C., & Tinelli, C. (2018). Satisfiability Modulo Theories. In *Handbook of Model Checking* (pp. 305–343). Springer.

Biscani, F., Fernando, I., Izzo, D., Kulal, S., 7ofNine, Čertík, O., & Pelteret, J.-P. (2018, May). *bluescarni/piranha: piranha 0.11.*

Bogomolov, S., Magazzeni, D., Minopoli, S., & Wehrle, M. (2015). PDDL+ Planning with Hybrid Automata: Foundations of Translating Must Behavior. In *Proceedings of the 25th International Conference on Automated Planning and Scheduling (ICAPS).*

Bogomolov, S., Magazzeni, D., Podelski, A., & Wehrle, M. (2014). Planning as Model Checking in Hybrid Domains. In *28th AAAI Conference on Artificial Intelligence.*

Bryce, D., Gao, S., Musliner, D. J., & Goldman, R. P. (2015). SMT-based Nonlinear PDDL+ Planning. In *Proceedings of the 29th AAAI Conference on Artificial Intelligence.*

Campion, J., Dent, C., Fox, M., Long, D., & Magazzeni, D. (2013). Challenge: Modelling Unit Commitment as a Planning Problem. In *Proceedings of the 23rd International Conference on Automated Planning and Scheduling (ICAPS).*

Cashmore, M., Fox, M., & Giunchiglia, E. (2012). Planning as Quantified Boolean Formulae. In *Proceedings of the 20th European Conference on Artificial Intelligence (ECAI).*

Cavada, R., Cimatti, A., Dorigatti, M., Griggio, A., Mariotti, A., Micheli, A., ... Tonetta, S. (2014). The nuXmv Symbolic Model Checker. In *Proceedings of Computer Aided Verification - 26th International Conference (CAV).*

Cenamor, I., Vallati, M., Chrpa, L., de la Rosa, T., & Fernndez, F. (n.d.). *TemPoRal: Temporal Portfolio Algorithm.* (IPC 2018 - Temporal Tracks)

Chen, Y., Xing, Z., & Zhang, W. (2007). Long-Distance Mutual Exclusion for Propositional Planning. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI)* (p. 1840-1845).

Cimatti, A., Griggio, A., Mover, S., & Tonetta, S. (2015). HyComp: An SMT-based Model Checker for Hybrid Systems. In *Proceedings of the European Joint Conferences on Theory and Practice of Software (ETAPS).*

Cimatti, A., Mover, S., & Tonetta, S. (2012). SMT-based Verification of Hybrid Systems. In *Proceedings of the 26th AAAI Conference on Artificial Intelligence.*

Coles, A., & Coles, A. (n.d.). *OPTIC.* (IPC 2018 - Temporal Tracks)

Coles, A. J., Coles, A. I., Fox, M., & Long, D. (2012). COLIN: Planning with Continuous Linear Numeric Change. *Journal of Artificial Intelligence Research*, *44*, 1–96.

Coles, Amanda and Fox, M and Long, D. (2013). A hybrid LP-RPG Heuristic for Modelling Numeric Resource Flows in Planning. *Journal of Artificial Intelligence Research*, *46*, 343–412.

Della Penna, G., Magazzeni, D., & Mercorio, F. (2012). A Universal Planning System for Hybrid Domains. *Applied Intelligence*, *36*(4), 932–959.

De Moura, L., & Bjørner, N. (2008). Z3: An Efficient SMT Solver. In *Proceedings of the 14th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS).*

Fernandez-Gonzalez, E., Williams, B., & Karpas, E. (2018). ScottyActivity: Mixed Discrete-Continuous Planning with Convex Optimization. *Journal of Artificial Intelligence Research*, *62*, 579–664.

Fox, M., & Long, D. (2002). PDDL+: Modeling continuous time dependent effects. In *Proceedings of the 3rd International NASA Workshop on Planning and Scheduling for Space* (Vol. 4, p. 34).

Fox, M., & Long, D. (2003). PDDL2.1: An Extension to PDDL for Expressing Temporal Planning Domains. *Journal of Artificial Intelligence Res. (JAIR)*, *20*, 61-124.

Fox, M., & Long, D. (2006). Modelling Mixed Discrete-Continuous Domains for Planning. *Journal of Artificial Intelligence Research (JAIR)*, *27*, 235–297.

Fox, M., Long, D., & Magazzeni, D. (2011). Automatic Construction of Efficient Multiple Battery Usage Policies. In *Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI).*

Fox, M., Long, D., & Magazzeni, D. (2012). Plan-based Policies for Efficient Multiple Battery Load Management. *Journal of Artificial Intelligence Research*, *44*, 335–382.

Furelos-Blanco, D., & Jonsson, A. (n.d.). *CP4TP: A Classical Planning for Temporal Planning Portfolio.* (IPC 2018 - Temporal Tracks)

Gao, S., Avigad, J., & Clarke, E. M. (2012). Delta-Complete Decision Procedures for Satisfiability over the Reals. In *Proceedings of the 6th International Joint Conference on Automated Reasoning (IJCAR).*

Gerevini, A., Saetti, A., & Serina, I. (2006). An Approach to Temporal Planning and Scheduling in Domains with Predictable Exogenous Events. *Journal of Artificial Intelligence Research*, *25*, 187–231.

Haslum, P., Ivankovic, F., Ramirez, M., Gordon, D., Thiébaux, S., Shivashankar, V., & Nau, D. S. (2018). Extending Classical Planning with State Constraints: Heuristics and Search for Optimal Planning. *Journal of Artificial Intelligence Research*, *62*, 373–431.

Henzinger, T. A. (1996). The Theory of Hybrid Automata. In *Proceedings of IEEE Symposium on Logic in Computer Science.*

Howey, R., & Long, D. (2003). VAL's Progress: The Automatic Validation Tool for PDDL2.1 used in the International Planning Competition. In *Proceddings of the ICAPS Workshop on IPC.*

Howey, R., Long, D., & Fox, M. (2004). VAL: Automatic Plan Validation, Continuous Effects and Mixed Initiative Planning Using PDDL. In *Proceedings of the 16th IEEE International Conference on Tools with Artificial Intelligence (ICTAI)* (pp. 294–301).

Huang, R., Chen, Y., & Zhang, W. (2012). SAS+ Planning as Satisfiability. *Journal of Artificial Intelligence Research*, *43*, 293–328.

Karaman, S., Walter, M. R., Perez, A., Frazzoli, E., & Teller, S. J. (2011). Anytime Motion Planning using the RRT. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA).*

Kautz, H., McAllester, D., & Selman, B. (1996). Encoding Plans in Propositional Logic. In *Proceedings of the 5th International Conference of Principles on Knowledge Representation and Reasoning (KR)* (pp. 374–384).

Kautz, H., & Selman, B. (1992). Planning as Satisfiability. In *Proceedings of the 10th European Conference on Artificial Intelligence (ECAI).*

Kautz, H., & Selman, B. (1996). Pushing the Envelope: Planning, Propositional Logic and Stochastic Search. In *Proceedings of the 13th National Conference on Artificial Intelligence and 8th Innovative Applications of Artificial Intelligence Conference (AAAI/IAAI).*

Kautz, H., & Selman, B. (1999). Unifying SAT-based and Graph-based Planning. In *Proceedings of the 16th International Joint Conference on Artificial Intelligence (IJCAI)* (pp. 318–325).

Kautz, H. A., Selman, B., & Hoffmann, J. (2006). SatPlan: Planning as Satisfiability. In *Abstracts of the 5th International Planning Competition.*

Lahijanian, M., Kavraki, L. E., & Vardi, M. Y. (2014). A Sampling-based Strategy Planner for Nondeterministic Hybrid Systems. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA).*

Léauté, T., & Williams, B. C. (2005). Coordinating Agile Systems through the Model-based Execution of Temporal Plans. In *Proceedings of the 20th National Conference on Artificial Intelligence (AAAI)* (p. 114).

Maly, M. R., Lahijanian, M., Kavraki, L. E., Kress-Gazit, H., & Vardi, M. Y. (2013). Iterative Temporal Motion Planning for Hybrid Systems in Partially Unknown Environments. In *Proceedings of the 16th International Workshop on Hybrid Systems: Computation and Control (HSCC).*

Mcallester, D., & Rosenblitt, D. (1991). Systematic Nonlinear Planning. In *Proceedings of the 9th National Conference on Artificial Intelligence (AAAI)* (Vol. 2, pp. 634–639).

Nabeshima, H., Iwanuma, K., & Inoue, K. (2002). Effective SAT Planning by Speculative Computation. In *AI 2002: Advances in Artificial Intelligence* (Vol. 2557). Springer.

Nathan Robinson and Charles Gretton and Duc Nghia Pham and Abdul Sattar. (2009). SAT-based Parallel Planning using a Split Representation of Actions. In *Proceedings of the 19th International Conference on Automated Planning and Scheduling (ICAPS).*

Oscar Sapena, E. O., Eliseo Marzal. (n.d.). *TFLAP: A Temporal Forward Partial-Order Planner.* (IPC 2018 - Temporal Tracks)

Penberthy, J. S., & Weld, D. S. (1994). Temporal Planning with Continuous Change. In *Proceedings of the 11th National Conference on Artificial Intelligence (AAAI).*

Piotrowski, W., Fox, M., Long, D., Magazzeni, D., & Mercorio, F. (2016). Heuristic Planning for Hybrid Systems. In *Proceedings of the 30th AAAI Conference on Artificial Intelligence (AAAI)* (pp. 4254–4255).

Plaku, E., Kavraki, L. E., & Vardi, M. Y. (2013). Falsification of LTL Safety Properties in Hybrid Systems. *International Journal on Software Tools for Technology Transfer*, *15*(4), 305–320.

Rankooh, M. F., & Ghassem-Sani, G. (2015). ITSAT: An Efficient SAT-based Temporal Planner. *Journal of Artificial Intelligence Research*, *53*, 541-632.

Rintanen, J. (1999). Constructing Conditional Plans by a Theorem-Prover. *Journal of Artificial Intelligence Research*, *10*, 323–352.

Rintanen, J. (2004). Evaluation Strategies for Planning as Satisfiability. In *Proceedings of the 16th European Conference on Artificial Intelligence (ECAI)* (pp. 682–687).

Rintanen, J. (2010a). Heuristics for Planning with SAT. In *Proceedings of the 16th International Conference on Principles and Practice of Constraint Programming (CP)* (pp. 414–428).

Rintanen, J. (2010b). Madagascar: Efficient Planning with SAT. In *the 7th International Planning Competition.*

Rintanen, J. (2017). Temporal Planning with Clock-Based SMT Encodings. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence (IJCAI)* (pp. 743–749).

Rintanen, J., Heljanko, K., & Niemelä, I. (2006). Planning as Satisfiability: Parallel Plans and Algorithms for Plan Search. *Artificial Intelligence*, *170*, 1031–1080.

Savas, E., Edelkamp, S., nad Derek Long, M. F., & Magazzeni, D. (n.d.). *Temporal-Numeric Planning with Infinite Domain Action Parameters.* (IPC 2018 - Temporal Tracks)

Savas, E., Fox, M., Long, D., & Magazzeni, D. (2016). Planning Using Actions with Control Parameters. In *Proceedings of the 22nd European Conference on Artificial Intelligence (ECAI)* (pp. 1185–1193).

Shin, J.-A., & Davis, E. (2005). Processes and continuous change in a sat-based planner. *Artificial Intelligence*, *166*(1).

SymPy. (2013). *Website.* (http://www.sympy.org/)

Tabuada, P., Pappas, G. J., & Lima, P. U. (2002). Composing Abstractions of Hybrid Systems. In *Proceedings of the 5th International Workshop on Hybrid Systems: Computation and Control (HSCC).*

Vallati, M., Magazzeni, D., Schutter, B. D., Chrpa, L., & Mccluskey, T. L. (2016). Efficient Macroscopic Urban Traffic Models for Reducing Congestion: A PDDL+ Planning Approach. In *Proceedings of the 30th AAAI Conference on Artificial Intelligence.*